НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра автоматики та управління в технічних системах

«На правах рукопису»
УДК _____

До захисту допущено:
Завідувач кафедри
_____ Олександр РОЛІК
«___»_____20__ р.

Магістерська дисертація

на здобуття ступеня магістра
за освітньо-професійною програмою
«Інтегровані інформаційні системи»
зі спеціальності 126 «Інформаційні системи та технології»
на тему: «Система захисту інформації веб-застосунку з оренди нерухомості»

Виконав (-ла):
студент (-ка) VI курсу, групи ІА-91мн
Земляной Олексій Олегович _____

Керівник:
Професор кафедри АУТС, д.т.н., проф.
Корнієнко Богдан Ярославович _____

Консультант: _____

Рецензент:
Доцент кафедри технічних та програмних
засобів автоматизації ІХФ, к.т.н., доцент
Ладієва Леся Ростиславівна _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2021 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра автоматизації і управління в технічних системах

Рівень вищої освіти – другий (магістерський)

Спеціальність – 126 «Інформаційні системи та технології»

Освітньо-наукова програма «Інтегровані інформаційні системи»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр РОЛІК
«___»_____20__ р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Земляному Олексію Олеговичу

1. Тема дисертації «Система захисту інформації веб-застосунку з оренди нерухомості», науковий керівник дисертації Корнієнко Богдан Ярославович, професор кафедри АУТС, д.т.н., затверджені наказом по університету від «_12_»_березня_ 2021 р. №_809-с_

2. Термін подання студентом дисертації ___11.05.2021_____

3. Об’єкт дослідження: системи захисту веб-застосунків.

4. Предмет дослідження: вразливості веб-застосунку з оренди нерухомості та методи їх знешкодження.

5. Перелік завдань, які потрібно розробити: Пошук та аналіз оптимальних підходів для вирішення проблеми. Аналіз переваг та недоліків наявних підходів. Опрацювання теоретичного матеріалу. Аналіз існуючих рішень, визначення недоліків систем та огляд можливих рішень для вирішення цих недоліків, вирішення проблеми актуальності написання нової системи. Опис пропонованої математичної моделі. Аналіз існуючих видів атак. Розробка програмного продукту.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу:

Діаграма прецедентів, діаграма уявлень, структура бази даних.

7. Орієнтовний перелік публікацій: 2 публікації

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

9. Дата видачі завдання 01.02.2021

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Пошук та аналіз оптимальних підходів для	12.03.2021	
	вирішення проблеми. Аналіз переваг		
	та недоліків наявних підходів.		
	Опрацювання теоретичного матеріалу.		
	Аналіз існуючих рішень, визначення		
	недоліків систем та огляд можливих		
	рішень для вирішення цих недоліків,		
	вирішення проблеми актуальності		
	написання нової системи.		
2	Опис пропонованої математичної	25.03.2021	
	моделі.		
3	Аналіз існуючих видів атак.	16.04.2021	
4	Розробка програмного продукту.	20.04.2021	
5	Оформлення текстової документації.	01.05.2021	
6	Підготовка ілюстративних матеріалів	11.05.2021	
7	Здача готової роботи.	11.05.2021	

Студент Олексій ЗЕМЛЯНОЙ

Науковий керівник Богдан КОРНІЄНКО

РЕФЕРАТ

Земляной О.О. Система захисту інформації веб-застосунку з оренди нерухомості. КПІ ім. Ігоря Сікорського, Київ, 2021.

Робота містить 105 с. тексту, 35 рисунків, 5 таблиць, посилання на 72 літературних джерела.

Ключові слова: система захисту, веб-застосунок, оренда та продаж нерухомості, «Білий список», C#, ASP.NET Core, Entity Framework Core, база даних, Microsoft Identity, AES.

Безпека веб-ресурсів являє собою важливий напрямок інформаційної безпеки. Кількість веб-ресурсів постійно зростає, разом з цим зростає як кількість приватної інформації, що зберігається на серверах віддаленого доступу, так і кількість атак на веб-ресурси з метою заволодіння такою інформацією. Подібного роду атаки потенційно можуть привести до великих негативних наслідків, економічних і репутаційних. Однак зацікавленість в здійсненні таких атак і захисту від них не обмежується сферою економіки. На сьогодні вразливість від атак може бути пов'язана з політичними мотивами, її використовують у гібридних війнах, вона стає чинником зростання терористичних загроз. Через це актуальність даної роботи полягає в зменшенні кібер-вразливості веб-застосунку.

Мета роботи – розробка системи захисту веб-застосунку з використанням сучасних технологій програмування та розроблення бази даних, яка буде стійкою до змін та сторонніх утручань, буде здатна запобігати неавторизованому доступу до веб-застосунку за допомогою низки методів злому / втручання.

Об’єкт дослідження – системи захисту веб-застосунків.

Предмет дослідження – вразливості веб-застосунку з оренди нерухомості та методи їх знешкодження.

ABSTRACT

Zemlianoi O.O. Information security system for real estate rental web application. Igor Sikorsky KPI, Kyiv, 2021.

The paper contains 105 pages, 35 figures, 5 tables, links to 72 literary sources.

Keywords: Information security system, web application, renting and selling real estate property, «White list», C#, ASP.NET Core, Entity Framework Core, database, Microsoft Identity, AES.

Web resource security is an important area of information security. The number of web resources is constantly growing, together with the growing amount of private information stored on remote access servers and with the number of attacks on web resources to get hold of such information. Such attacks can potentially have major negative consequences, both economic and reputational. However, the interest in carrying out such attacks and protecting against them is not limited to the sphere of economy. Nowadays, vulnerability to attacks can be linked to political motives, it is used in hybrid wars, it is becoming a factor in the growth of terrorist threats. Therefore, the relevance of this research is to decrease the cyber vulnerability of a web application.

The aim of the master's thesis is to develop the security system for the web application using modern programming and data base development technologies so that the system could be resistant to change and outside interference and be able to prevent unauthorized access to the web application with the help of a number of methods of hacking / intervention.

The object of the research is web application protection systems.

The subject of the research is vulnerabilities of web real estate rental applications and methods of their resolving.

ЗМІСТ

ВСТУП

Безпека веб-ресурсів являє собою важливий напрямок інформаційної безпеки. Кількість веб-ресурсів постійно зростає, разом з цим зростає як кількість приватної інформації, що зберігається на серверах віддаленого доступу, так і кількість атак на веб-ресурси з метою заволодіння такою інформацією. Подібного роду атаки потенційно можуть привести до великих негативних наслідків, економічних і репутаційних Однак зацікавленість в здійсненні таких атак і захисту від них не обмежується сферою економіки. На сьогодні, вразливість від атак також може бути пов'язана з політичними мотивами, її використовують у гібридних війнах, вона стає чинником зростання терористичних загроз.

При збільшенні залежності компаній від напрямку IT-технологій, гостро постає питання належного забезпечення інформаційної безпеки. Ключовим заходом забезпечення безпеки в компаніях можна назвати тестування продуктів на проникнення. Такий вид тестування дозволяє захистити продукти від основних загроз інформаційної безпеки, таких як несанкціонований доступ та інших[1].

Аналіз повідомлень засобів масової інформації та даних, які надає команда аварійних подій в Україні (CERT-UA), дає можливість зробити висновок, що протягом останніх років втручання груп організованих зловмисників в роботу комп'ютерних систем багатьох підприємств та установ викликало блокування роботи цих систем, та як наслідок, призвело до матеріальних та репутаційних збитків[1].

Необхідно визнати, що на сьогоднішній день законодавство України в сфері захисту інформації певною мірою є застарілим. Крім того, в країні не існує положень для врегулювання сучасних проблем інформаційної безпеки. Ще одним важливим фактором є те, що відсутня єдина методика та набор механізмів для захисту від проникнення зловмисників[2]. Саме через це рівень захисту комп'ютерних систем не відповідає актуальним загрозам.

У зв'язку з цим пропонується розробити систему, що буде захищена від

набору можливих варіантів проникнення, втручання або неавторизованого доступу до інформації, який підходитиме для українських стандартів [2].

Мета роботи - розробка системи захисту веб-застосунку з використанням сучасних технологій програмування та розроблення бази даних, яка буде стійкою до змін та сторонніх утручань, буде здатна запобігати неавторизованому доступу до веб-застосунку за допомогою низки методів злому\втручання.

Об’єкт дослідження - системи захисту веб-застосунків.

Предмет дослідження - вразливості веб-застосунку з оренди нерухомості та методи їх знешкодження.

Наукова новизна одержаних результатів - агрегація існуючих методів захисту на прикладі веб-застосунку, практичне впровадження моделі «Білого списку».

Практичне значення одержаних результатів полягає у зменшенні кібер-вразливості веб-застосунку.

1 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

Сьогодні веб-уразливість переважує будь-які інші проблеми інформаційної безпеки. Більшість зовнішніх атак на корпоративні інформаційні системи націлені на вразливість веб-додатків.

Багато компаній, навіть із незвичних сегментів ринку, повністю переходять в Інтернет. І поширення онлайн-платежів лише посилює цю тенденцію. Досвід показав, що там, де здійснюються продажі в Інтернеті, використання спеціалізованого захисного обладнання є справді критичним і необхідним.

Серед основних загроз на рівні використання програмних вразливостей зловмисники активно використовують фішинг-інструменти, побудовані на людських помилках. Нам нікуди втікти від цього, але фахівці з інформаційної безпеки також не сидять склавши руки.

1.1 Аналіз актуальних атак на веб-ресурси

Аналіз здійснено із використанням існуючих статистичних даних за 2015 рік [3]. Наведено не тільки опис атак і відсоток веб-ресурсів, до яких вони можуть бути застосовані, автором також запропоновані адекватні методи протидії таким атакам. Для наочності результати зведено в таблицю.

Найбільш популярною атакою є «Insufficient transport layer protection» – отримання даних під час передавання. Дана атака може бути виконана для 70% ресурсів. Для виключення можливості проведення таких атак достатньо використовувати протокол HTTPS.

Ще одною формою атаки є «Витік інформації» («Information leakage»). Дану атаку можна виконати на 56% ресурсів. Витік інформації з додатків виникає в результаті відмови або неправильної роботи програми, а також у разі порушення її логіки. Для виключення можливості проведення атаки необхідно ретельно тестувати програмну частину ресурсу, проводити перевірку повідомлень на стороні сервера,

моніторинг оповіщень про помилки.

Таблиця 1 – Класифікація видів атак

№ за/п	Вид атаки	Вразливість веб-ресурсів, %	Протидія
1	Insufficient transport layer protection	70 %	Використання протоколу HTTPS.
2	Information leakage	56 %	Тестування програмної частини ресурсу, перевірка повідомлень на стороні сервера, моніторинг оповіщень про помилки
3	Cross-site scripting	47 %	Очищення та валідація вхідних даних
4	Brute force	29 %	Використання паролів високої складності, налаштування сервера на аналіз вхідних запитів
5	Content spoofing	26 %	Відмовитися від використання фреймів і не передавати в параметрах абсолютні або локальні шляхи до файлів
6	Cross-site request forgery	24 %	Перевірка вхідних даних з форм
7	URL redirector abuse	16 %	Валідація вхідних даних
8	Predictable resource location	15 %	Контроль доступу до файлів сервера

Атаку «Cross-site scripting» (міжсайтове використання сценаріїв) можливо виконати на 47% ресурсів. Атака може містити JavaScript-код, що обробиться браузером жертви. Вразливості, що мають такий характер, називають HTML-ін'єкціями, через то, що механізм їхнього впровадження дуже схожий із SQL-ін'єкціями, але на відміну від останніх, впроваджуваний код виконується в браузері користувача. Для захисту від цього виду атак необхідно проводити очищення та валідацію вхідних даних.

Генерацію великої кількості запитів або підбір паролів («Brute force») можливо виконати на 29% ресурсів. Для захисту необхідно забезпечити використання паролів високої складності, налаштування сервера на аналіз вхідних запитів.

Атака «Content spoofing» (підмінна даних через заміну контенту сторінок)

можлива для 26% ресурсів. Використовуючи цю техніку, зловмисник змушує користувача повірити, що сторінка згенерована веб-сервером, а не передана із зовнішнього джерела. Для захисту від даного виду атак потрібно відмовитися від використання фреймів і, найголовніше, ніколи не передавати в параметрах абсолютні або локальні шляхи до файлів.

Вид атак на відвідувачів веб-сайтів, який використовує недоліки протоколу HTTP, називається «Cross-site request forgery». Коли жертва авторизується на сайті, створений зловмисником, від його імені таємно відправляється запит на сторонній сервер (наприклад, на сервер сторонньої платіжної системи), і на ньому здійснюють шкідливу операцію (наприклад, переказ на сторонній рахунок, що належить зловмиснику). Дану атаку можливо виконати на 24% ресурсів. Для захисту необхідно проводити перевірку вхідних даних з форм, наприклад шляхом додавання унікального доданка.

Переправлення на інші сайти через підміну початкових посилань («URL redirector abuse»), також як і багато інших перерахованих вище, є різновидом помилок перевірки вхідних даних і є можливим на 16% ресурсів. Вирішенням є валідація вхідних даних.

Ще однією популярною атакою є «Predictable resource location», тобто знаходження прихованого функціоналу та даних. Така атака доступна на 15% ресурсів і вирішується шляхом контролю доступу до файлів сервера.

З кожним роком статистика атак змінюється. Так, у 2014 році найпопулярнішою формою була «Cross-site scripting», а в 2013 – витік інформації («Information leakage»). Виходячи з наведених даних, можна зробити висновки про те, що для захисту від більшості популярних видів атак достатньо належним чином перевіряти вхідні дані. Також рекомендовано використовувати шифрований протокол HTTPS та будувати програмний додаток ресурсу на одному з відомих програмних каркасів (Frame-work), в якому вбудовані механізми перевірки, шифрування та валідація вхідних даних

Також варто зауважити, що в даному дослідженні не розглянуті атаки на мережеві служби, наприклад DoS та DDoS, найкращим методом захисту від яких є

використання хмарних технологій і перевірених конфігурацій серверів.

1.2 WAF (Web Application Firewall)

Суть Web Application Firewall полягає у захисті WEB ресурсів компанії, які виставлені назовні. Таке рішення дозволяє захищати клієнтів від багатьох видів атак. Серед них є як SQL і PHP-ін'єкції, так і cross-site scripting (XSS), підбір паролів (broot force та інші), від експлуатації вразливостей zero-day в веб-аплікаціях, DDoS-атаки рівня додатків і атаки на інші ресурсомісткі сторінки. Деякі виробники мають цілий ряд попередньо налаштованих протоколів доступу для захисту додатків, таких як Sharepoint, Oracle E-Business Financials, Microsoft Outlook Web Access, Lotus Domino Mail Server та інших.

Виникає питання, які є можливості цих рішень. Чим вони відрізняються від традиційних методів захисту, таких як IPS iNGFW? По-перше, потрібно розуміти, що WAF — це вузькоспеціалізований пристрій, він активно контролює тільки HTTP / HTTPS протоколи. Існує також такий клас атак, як CSRF, і він націлений на клієнтів веб-додатків. Трафік такої атаки зупиняється на клієнтському фаєрволі. Може здаватися, що захистити користувача досить складно, але це можливо.

Розглянемо такий сценарій атаки: користувач переходить на веб-сайт банку, відбувається автентифікація, а потім на іншій вкладці браузера відкривається заражений ресурс. За допомогою JavaScript заражений ресурс в іншому вікні може таємно запитувати переказ грошей від користувача, і браузер замінить усі необхідні параметри автентифікації для фінансової операції, оскільки сеанс користувача з банком ще не закінчився. В описаній ситуації ми маємо вразливість алгоритму автентифікації в банківському програмному забезпеченні. Якби для кожної форми, що міститься на сторінці веб-сайту, було створено унікальний маркер, проблем не було б.

На жаль, розробники програмного забезпечення майже ніколи не піклуються про безпеку так глибоко. Однак деякі WAF можуть самостійно реалізувати такий

захист у веб-формах і таким чином захистити клієнта, а точніше його запити, дані, URL-адреси та файли cookie.

1.3 OpenVAS

OpenVAS – це повнофункціональний сканер вразливостей. Його функції включають неавтентифіковане тестування, автентифіковане тестування, різні високоякісні та низькорівневі Інтернет та промислові протоколи, налаштування продуктивності для широкомасштабного сканування та потужну внутрішню мову програмування для реалізації будь-якого типу тесту на вразливість.

Сканер супроводжується стрічкою тестів на вразливість з великою історією та щоденними оновленнями. Цей потік спільноти Greenbone включає понад 80 000 тестів на вразливість.

Сканер розробляється та підтримується компанією Greenbone Networks з 2009 року. Роботи надаються спільноті з відкритим кодом під загальною публічною ліцензією GNU (GNU GPL).

Greenbone розробляє OpenVAS як частину свого комерційного сімейства продуктів управління вразливістю, Greenbone Security Manager (GSM). OpenVAS – це один із елементів більшої архітектури. У поєднанні з плагінами з відкритим кодом він утворює рішення для управління вразливістю Greenbone. Виходячи з цього, пристрої GSM використовують ширший канал, який охоплює потреби підприємства, GVM з додатковими функціями, управлінням пристроями та угодою про рівень обслуговування.

1.4 Аналіз системи CVSS 3.0.

Спільна система оцінки вразливості (CVSS) є відкритою основою для передачі характеристик та серйозності програмних вразливостей. CVSS складається з трьох метричних груп: базової, тимчасової та екологічної. Базова група представляє

внутрішні якості вразливості, тимчасова група відображає характеристики вразливості, що змінюються з часом, а екологічна група представляє характеристики вразливості, які є унікальними для середовища користувача. Базові показники дають оцінку в діапазоні від 0 до 10, яку потім можна змінити, оцінивши часові та екологічні показники оцінки. Оцінка CVSS також представлена у вигляді векторного рядка, стислого текстового подання значень, що використовуються для відображення оцінок. [4]

Базові показники описують складність використання вразливості та потенційну шкоду конфіденційності, цілісності та доступності інформації.

Метрики :

- вектор атаки – це міра віддаленості можливого атакуючого актора від вразливого об'єкту. Метрика може приймати наступні параметри: Network (N), Adjacent Network (A), Local (L), Physical (P)
- межі експлуатації – визначає чи відрізняються експлуатований вид компонентів, що підлягають атаці, тобто визначає чи експлуатація уразливості порушує цілісність, конфіденційність і доступність усілякого компонента системи. Метрика може приймати наступні параметри: Unchanged (U), Changed (C)
- необхідність взаємодії з користувачем – визначає чи потрібні для успішного проведення атаки будь-які дії з боку користувача системи, що підлягає атаці. Метрика може приймати наступні параметри: None (N), Required (R)
- автентифікація / необхідний рівень привілеїв – визначає чи потрібна автентифікація для виконання атаки, і, якщо потрібна, то яка саме. Можливі значення метрики: High (H), Low (L), None (N)
- метрики впливу – визначає міру важливості на конфіденційність, доступність і цілісність компонента, що атакується. Метрика може приймати наступні параметри: Medium (M), High (H)
- складність експлуатації уразливості – це характеристика складності

технічних засобів для проведення атаки. Складність впровадження залежить напряму від умов, що вимагає система. Метрика може приймати наступні параметри: Low (L), High (H) [5]

Відповідно до показника оцінки є таблиця відповідності рейтинга, ці значення наведено в таблиці 1.1.

Таблиця 1.2 – Значення рейтингу вразливості

Немає	0,0
Низький	0,1 – 3,9
Середній	4,0 – 6,9
Високий	7,0 – 8,9
Критичний	9,0 – 10,0

Бачимо, що мається механізм оцінки значення критичної вразливості в числовому еквіваленті. У цій роботі я буду використовувати оцінку CVSS 3.0, щоб провести аналіз актуальних методик. За допомогою розробленої методики можна проводити тестування системи на наявність вразливостей тільки із критичними значеннями. [6]

1.5 Проведення порівняльного аналізу існуючих методологій тестування інформаційної безпеки

Для проведення порівняльного аналізу існуючих методологій тестування інформаційної безпеки на даний момент найбільш розповсюдженими методологіями тестування на зловмисне проникнення є:

– BSI – Study A Penetration Testing Model.

– OWASP Testing Guide;

– Information Systems Security Assessment Framework (ISSAF);

– The Open Source Security Testing Methodology Manual (OSSTMM);

– The National Institute of Standards and Technology (NIST) Special Publication 800-115;

– Penetration Testing Execution Standard (PTES);

– OWASP Testing Guide;

Аналіз методології

Посібник з методології тестування безпеки з відкритим кодом (OSSTMM) – це досить формалізований та добре структурований документ для мережевого тестування. Документ має так звану «Карту безпеки» – візуальний індикатор безпеки. На карті вказані основні сфери безпеки, які включають набори елементів, які необхідно перевірити на відповідність методології. Документ містить підпункт «Методологія» / «Тестування технологій безпеки в Інтернеті» / «Огляд мережі» / «Тестування брандмауер»", де перелічується очікувана інформація, яку зловмисник може отримати в результаті успішної атаки або відсутності необхідного функцію в інструментах безпеки. Він також описує конкретні правильні реакції мережі на атаки та їх присутність, наприклад, вимірювання часу відгуку на пакет або перевірку втрат пакетів на шляху до пункту призначення. [7]

Переваги такої методики:

– детально описані процедури щодо підготовки до тестування;

– детальний опис основних термінів та понять у галузі інформаційної безпеки;

– детальний опис методів та підходів до тестування.

Мінусами цієї методики можна вважати :

– формалізованність;

– відсутність додаткового опису щодо вимог;

– те, що безкоштовна версія посібника OSSTMM V3 була опублікована у 2010 році і частково застаріла. Актуальна версія доступна лише для платних

учасників;

– відсутній опис інструментів, які необхідно використати для цього.

Методологія NIST Special Publications 800-115 Technical Guide toInformation Security Testing and Assessment. Створено та підтримується NIST і визначає щонайменше 3 фази оцінки інформаційної безпеки: планування, впровадження, післяопераційна (аналіз даних, виявлення причин уразливості, розробка рекомендацій щодо вразливості та розробка звітів). У розділі «Методи оцінки цільової вразливості» описуються тести на проникнення як один із методів, а саме фази та логістика тестів.

Тести на проникнення, з їх стандартними можливостями, можна застосувати для визначення:

– як система реагує на існуючі моделі атак;

– настільки складна система для атаки;

– заходи протидії, які спрямовані на послаблення адреси системи;

– здатність захисту системи на атаки та забезпечення реакції на них.

Переваги цього документу полягають в тому, що:

– він включає загальний опис методів перевірки комп'ютерних систем, а також короткий опис даних методів (наприклад огляд мережі, перевірка налаштувань системи, перевірка файлів журналів, перевірка цілісності файлів, сканування вразливостей мереж, тощо);

– приводяться посилання на програмне забезпечення, яке можна використовувати для тестування;

– він включає в себе посилання на інші методології та необхідні нормативні документи.

Серед недоліків методу необхідно відзначити те, що:

– даний документ було розроблено в 2008 році, і на сьогодні він не відповідає сучасному стану інформаційних технологій через те, що методи проникнення в комп’ютерні мережі з кожним роком вдосконалюються.

Посібник з тестування методології OWASP (Open Web ApplicationSecurity Project). OWASP (Open Web Application Security Project) – це інтернаціональна відкрита спільнота, яка зосереджується на поліпшенні якості коду ПО. До OWASP

може долучитися довільний бажаючий, і всі їх матеріали розповсюджуються вільно. Посібник з тестування OWASP є більш широкою методологією у порівнянні з іншими, оскільки він надає вказівки не тільки щодо тестів на проникнення, але й аналізу веб-додатків загалом (наприклад, вихідного коду), оскільки ця методика зосереджена на виявленні вразливостей в веб-застосунках.

Перевага цього документу полягає в тому, що:

- керівництво OWASP надає усю необхідну інформацію щодо кожного етапу життєвого циклу розробки захищеного програмного забезпечення;
- це сама популярна та повна колекція засобів щодо тестування безпеки веб-додатків, серед актуальних.

Мінусами методики вважається те, що:

- якщо веб-сайт або веб-додатки компанії не є критичними з точки зору бізнесу, то заходи щодо тестування на проникнення із застосуванням цієї методології можуть бути не доцільним.

Методологія PTES - Penetration Testing Execution Standard - TechnicalGuidelines. Стандарт, розроблений для об'єднання бізнес вимог та можливостей відділів безпеки, а також щодо масштабування тестів на проникнення. Перший підготовчий етап детально розглядає встановлення каналу комунікації, правила взаємодії та контролю, конкретні способи реагування та моніторингу існуючих інцидентів.

Слід окремо виділити наступні етапи:

- збір інформації;
- імітація загроз;
- аналізу вразливостей системи;
- експлуатація;
- забезпечення та реалізація контрзаходів та виявлення найлегшого шляху атаки;
- подальший аналіз інфраструктури, заходи щодо проникнення в інфраструктуру, зачистка після тестів та тестування живучості;
- формування структури звітів, які включають результати тестування.

Перевага даної методології:

– системи керування, які містять детальну технічну інформацію щодо інструментів та команд для кожного етапу тестування.

Серед мінусів методики можна відмітити те, що:

– не приділяється достатньої уваги методам соціальної інженерії.

Методологія ISSAF - Information System Security Assessment Framework, була розроблена для внутрішніх перевірок. Методологія розглядає широкий спектр питань, пов'язаних з інформаційною безпекою. У них присутні частини в яких розглядаються оцінка безпеки міжмережевих екранів маршрутизаторів, антивірусних систем та багато інших. Методологія ISSAF надає змогу моделювання вимог для довнутрішніх заходів безпеки та направлення для оцінки безпеки комп'ютерних мереж, додатків та систем. Методологія більш глибоко, ніж PTES, фокусується на питання перевірки безпеки комп'ютерних систем та визначає, як використовувати різні інструменти. Система оцінювання безпеки інформаційних систем(ISSAF) має дві частини: управлінську та технічну. Управлінська містить загальні рекомендації та використовується для успішного процесу тестування. Технічна включає в себе набір правил та процедур для створення об'єктивного процесу оцінки безпеки [8].

Плюси методології ISSAF:

– дозволяє подолати розрив між управлінською та технічними частинами тестування захисту та реалізує засоби контролю, які необхідні для результативності управління обома сторонами.

Мінуси методології:

– методологія застаріла.

Методологія BDI – розглядається модель Penetration Testing Model, що розроблена підрозділом «Federal Office for Information Security» (FOIS) з Германії. У методології розглянуто проведення випробовування безпеки системи. Тут розглядається не тільки сама методологія тестування, але також й вимоги, що необхідно виконати для проведення тестів, в яких проводиться класифікація тестів на міцність і визначення критеріїв.

Плюси методології BDI :

– методика прискіплива, намагається перевірити всі наявні аспекти на міцність, такі як: технічні, правові, організаційні;

– додатки містять опис ПЗ, які можна використати у тестуванні об’єктів описаних в методології.

1.6 Висновки до розділу 1

У даному розділі було розглянуто основні види вразливостей веб-застосунків та описано найпопулярніші готові сервіси для впровадження відповідного захисту. Проте вони всі є іноземними розробками. Як можна бачити, в українському просторі сформовано перелік необхідних методик та заходів запобігання кібер-вразливості веб-додатків, проте відсутнє готове рішення.

Провівши аналіз проблеми створення мережевого програмного забезпечення, можна зробити висновок, що ця проблема актуальна і по сьогодні. Вона також має спільні риси з загальною концепцією створення захищеного прикладного програмного забезпечення. Проблема, що розглядається, залежить від механізмів захисту веб-шаблонів, на яких будується програмне забезпечення, базі даних, що використовується, захищеності сервера в цілому, і найбільше за все від професійності розробників програмного продукту.

В даній роботі буде дороблено бєк-єнд частину веб-застосунку, котра включає в себе функціонал запобігання методам злому, які зустрічаються найчастіше.

2 ОПИС МОДЕЛІ

У наступному розділі буде розглянуто концепцію «Білого списку», використання «Білого списку» в IT технологіях, створення методики розробки захищених веб-програм на основі математичної моделі «Білого списку», а також приклад використання даної методики.

2.1 Забезпечення безпеки у веб-застосунках з використанням методу «Білий список»

Методологія «Білого списку» загально відома і застосовується досить давно у різних сферах ще до появи інформаційних цифрових технологій. Методиці «Білий список» можна дати наступне визначення: «Білий список» — це практика ідентифікації суб'єктів, яким надаються визначені привілеї, послуги, мобільність, а також рівень доступу. Суб'єкти в даному списку будуть прийняті, схвалені та будуть признані. «Білий список» — це протилежність «Чорних списків». «Чорний список» - це практика ідентифікації осіб, які заперечуються та не визнаються.

Досить часто «Білий список» використовують для захисту від спаму або шкідливого програмного забезпечення, налаштування мережевого доступу, а також під час веріфікації будь-яких об’єктів, а також вхідних та вихідних даних.

Наприклад, потрібно налаштувати доступ до бази даних що знаходиться на віддаленому веб ресурсі. Для підвищення рівня безпеки потрібно додати в налаштуваннях підключення IP адресу у «Білий список», яка в подальшому буде використовуватися для підключення до бази даних. Такий простий механізм виключить можливість підключення нашої бази даних з інших комп’ютерів окрім того, IP адреса якого знаходиться у «Білому списку».

«Білий список» можна визначити як деяку множину елементів (наприклад, список електронних адрес, програм, або інші дані) яким надається спеціальна привілея на виконання або обробку даних, чи доступ до якогось ресурсу.

Протилежним до такого методу є метод «Чорного списку» – множина елементів у яких такі права відібрані. «Білий список» як і «Чорний список» часто використовуються в різних механізмах захисту мережевих систем та інфраструктур.

2.2 Ідея використання білого списку під час безпечної розробки

З теоретичної точки зору, використання моделі «Білий список» для розробки веб-додатків дозволяє застосунку бути нечутливим до деяких вразливостей, що притаманні більшості популярних рішень існуючих веб-каркасів.

Головна мета – продемонструвати, що створений веб-додаток відповідає концепції «Білий список» і використаний метод здатний бути невразливим до деяких вразливостей, а саме A4 (відсутність рівнів функцій контролю доступу) і A7 (небезпечні прямі посилання на об'єкти) з OWASP TOP 10.

2.3 Опис моделі "Білого списку" щодо розробки захищених веб-аплікацій

Сучасні веб-фреймворки можуть запобігати більшості відомих вразливостей веб-додатків, наприклад, SQL-ін’єкції та XXS. Але нажаль вони не зможуть запобігти специфічним вразливостям, які можуть бути частиною конкретного програмного забезпечення, яке розробляється цим фреймворком, відповідальність та запобігання таким вразливостям залишається за розробниками. Оскільки хакери знаходять постійно вдосконалюють методи експлуатації вразливостей систем, розробникам потрібно постійно виділяти все більше ресурсів на впровадження заходів щодо забезпечення захисту систем. В теорії, методика «Білого списку» розробки веб-аплікацій дозволяє розробникам підвищувати рівень безпеки завдяки виявленню непередбачуваної та небезпечної поведінки веб-аплікації, за допомогою припинення її роботи або перенаправляючи систему на безпечну роботу.

Розбіжність між очікуваними та реальними поведінками веб-застосунку може бути оповіщенням атаки на веб-застосунок. Було використано OWASP4 та 7

вразливостей: прямі посилання на об’єкти, що є небезпечними, відсутність можливості контролювати доступ функцій. Необхідно реалізувати захист веб-застосунку шляхом передбачення дозволених операцій, такий підхід можна використати частково, абстрактно.

Для спрощення моніторингу безпеки, можна розробити веб-застосунок, за допомогою взаємодій, що є дозволеними для компонентів застосунку. Якщо послідовність дій не була введена в специфікацію, то такі дії будуть відхилені логікою застосунку, та переправленні. Необхідним є визначення та створення дозволених взаємодій компонентів застосунку. Дані правила можуть керувати HTTP запитами та відповідями на нього, що обробляє веб-застосунок.

Визначення послідовностей, що будуть входити до «Білого списку» необхідно розробляти на етапі проектування. Для успішної роботи «Білий список» необхідно створювати динамічно під час розробки, за допомогою засобів моніторингу поведінки та виконання застосунку. Для з’ясування, чи буде покращена робота безпеки веб-застосунку, можна використати статичний «Білий список», який можна визначити на етапі проектування. Створення «Білого списку» можлива при реалізації веб-застосунку з використанням будь-якого фреймворку на основі MVC або Web API через один з варіантів роботи таких фреймворків – це механізм, що працює у вигляді посередника та є відповідальною за перехоплення HTTP запиту та видачі відповіді, що задається моделлю з «Білого списку».

2.4 Формальне визначення моделі білого списку розробки безпечних веб-застосунків

Визначимо білий список як набір чотирьох множин {A, B, W, S}, де:

C множина елементів {u, a_1, a_2,…,a_n}, де a_1, a_2,…,a_n – складові компонент які входять в межі системи, а u компоненти які за межею системи.

D множина елементів {b_1, b_2,…,a_n}, де b_1, b_2,…,a_n стани компонентів.

Кожна комірка матриці розмірності |A| X |A| містить унікальну підмножину x,

{x: x ⊆ B}. Якщо оцінка станів x повертає 1, то упорядкована пара переходу стану з $a_{початковий(origin)}$ до $a_{кінцевий(destination)}$ додається до множини W.

W множина впорядкованих пар $\{(a_o, a_b) : a_o, a_b \in a\}$ кожна пара представляє собою перехід з початкового компоненту c_0 до кінцевого a_b

S матриця розмірності |A| X |A| $S_{ao, ab} = a_{безпечний(safe)}$ визначає безпечні компоненти $\{a_s : a_s \in A\}$ де a_d не може слідувати з a_o.

Перехід з однієї складової до іншої керується функцією переходу де перехід з $a_{початковий}$ до $a_{кінцевий}$ відбувається тоді і тільки тоді якщо $(a_o, a_b) \in W$, інакше функція переходу визивається через $(a_o, S_{ao, ab})$.

$$T(a_0, a_b) = \begin{cases} a_b, \text{якщо } (a_0, a_b) \in W \text{ інакше,} \\ T(a_0, S_{a_0\, a_b}) \end{cases}$$

Завдяки «Білому списку» можна робити декілька визначених операцій. Вони поділені на дві категорії, в залежності від того, в якому місці роботи та коли вони можуть бути застосовані. Для розробки будуть використані наступні операції на множинах:

- виникнення (a_o, a_b) у множині W: додавання впорядкованої пари до множини;
- прибирання (a_o, a_b) з множини W: видалення впорядкованої пари з множини;
- ввід $\{b_x\}$ до множини B: додавання станів b_x до множини B;
- видалення $\{b_x\}$ з множини B: видалення станів b_x з множини B;
- додавання $\{b_x\}$ до підмножини x множини $W_{ao,ab}$;
- видалення $\{b_x\}$ з підмножини x множини $W_{ao,ab}$;
- ввід $\{a_s\}$ у комірку матриці $S_{ao,ab}$;
- оновлення $\{a_s\}$ у комірці матриці.

Операції білого списку які дозволені при виконані:

– вичислення $T(a_{o,} a_b)$;

– варифікація $a_o \rightarrow a_b$:

c_d може слідувати до a_o якщо дане твердження хибне, то усі стани які належать підмножині x у $W_{ao,ab}$ будуть повертати істину.

Інакше перехід до безпечного стану a_s.

Множина усіх компонентів A і відношень W можуть бути представлені у вигляді двійкової матриці, де 1 відповідає дозволеному переходу стану, а 0 - забороненому. В залежності від значень підмножини станів, в ячейках матриці можуть бути вписані значення 1 та 0. Обачні переходи станів записано до матриці S, для сценаріїв, коли перехід стану з a_o до a_b заборонено.

2.5 Кроки впровадження "Білого списку"

По-перше, необхідно ідентифікувати дозволену поведінку веб-аплікації за допомогою створення діаграм які будуть відображати те, як поводить себе програма. Діаграма повинна ілюструвати усі дозволені взаємодії між компонентами програми. Необхідно визначити кожну операцію та ідентифікувати систему переходів станів (a_o, a_b) і помістити їх у відповідні частини матриці W. Також необхідно визначити захищені компоненти та переходи, та у разі повертання невірного значення перевірки дозволеність переходу до нового стану. Захищені компоненти a_s слід помістити у частини матриці, які відповідають (a_o, a_b) у матриці W до матриці S. Як наслідок, на цьому етапі розробки ми маємо компоненти переходу станів які записані до матриці W та приймають значення 1 або 0 в залежності від дозволеності переходу до станів. Ми можемо назвати такі представлення як M, де |A| X |A| = M де $M_{ao,ab} = 1$ якщо (a_o, a_b) є W і $M_{ao,ab} = 0$ якщо (a_o, a_b) не належить моделі W. Модель "Білого списку" доповнюється та змінюється в залежності від етапу розробки. Слід відмітити що, стани в компонентах B повинні бути простими, а не комплексними.

Необхідно виконати такі пункти для створення моделі «Білого списку»:

– створити схему, яка позначає заапрувлену безпечну поведінку веб –

додатку;

- означити підмножини станів для проведення аналізу і позначання дозволених кроків між станами і схеми відношень компонентів застосунку;
- всі підмножини переходів станів a_o,a_b будуть зіставлені до відповідних комірок у матриці |A|x|A|;
- ідентифікувати безпечні компоненти a_s і розмістити їх у комірки які відповідають (a_o,a_b) у матриці W до матриці S;
- позначити 1 або 0 для кожної підмножини переходу станів;
- $M_{ao,ab}$ = 1 якщо (a_o, a_b) є W і $M_{ao,cb}$ = 0 якщо (a_o, a_b) не належить;
- процес побудови веб-застосунку повинен інтегрувати до себе підхід «Білого списку», інтеграція повинна бути успішною для всіх етапів розробки додатку

2.6 Приклад застосування

Для наглядного прикладу застосування мтеоду "Білого списку" при розробці веб-програм, приведемо наступний приклад. Припустимо що є програмне забезпечення, яке вимагає обов’язкову аутентифікацію користувача для подальшого його використання.

Нехай програма дозволяє 3 спроби аутентифікації. Якщо користувач не пройде аутентифікацію за 3 спроби, програма заблокує доступ даному користувачу. Якщо користувач успішно пройде аутентифікацію - програма перенаправить користувача на його персональну сторінку. З цього моменту користувач матиме змогу редагувати свої персональні дані, а також використовувати іншими можливості програмного продукту, написклад зв"язатися з іншими користувачами. Користувач також має змогу вийти зі свого аккаунту в будь-який зручний для його час.

Зелені стрілки – операції, підтверджені «Білим списком». Червоні відповідно не підтверджені.

Модель програмного забезпечення даного прикладу складається з 5 складових. A = {u, a_1, a_2, a_3, a_4, a_5}. U представляє собою компоненти, які знаходяться за границею системи, і включається до A за для повноти. B - глобальна множина станів системи. Вона визначає наступні стани:

- b1: користувач анонімний.
- b2: коистувач авторизований.
- b3: час існування сесії валіний.
- b4: попереднє представлення.
- b5: представлення підпослідовності
- b6: спроби аутентифікації < 3.

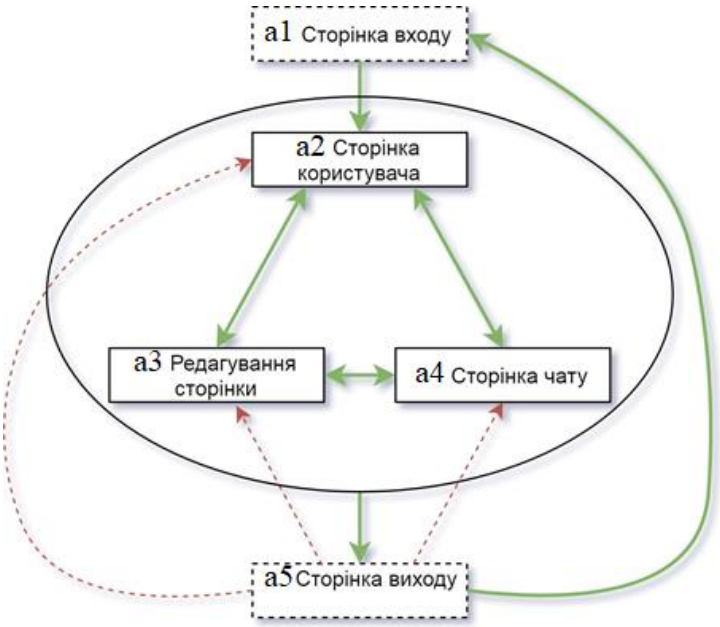

Рисунок 2.1 – Приклад діаграми яка відображає поведінку програми

Для всіх комірок матриці «Білого списку» існує елемент з підмножини B. «Білій список», що буде розглянуто далі, співставляє підмножину дозволених переходів станів з a_1, a_2 і іншу підмножину, що відповідає забороненим крокам з a_5 до a_4. Для дозволеного переходу з a_1 до a_2, підмножина станів: x = {b_2, b_3, b_4 = login view, b_5 =

user portal view, b_6}. Усі стани у x мають повертати істину. Для недозволених переходів з a_5 до a_4, підмножина станів x = {b_2, b_3, b_4 = edit profile view або user portal view, b_6}.

Таблиця 2.1 – матриця W впорядкованих пар станів переходу компонентів

	u	a1	a1	a3	a4	a5
U	-	-	-	-	-	-
a1	-	-	x = {b_2, b_3, b_4 = login view, b_5 = user portal view, b_6}	-	-	-
a2	-	-	-	-	-	-
a3	-	-	-	-	-	-
a4	-	-	-	-	x = {b_2, b_3, b_4 = edit profile view або user portal view, b_6}	-
a5	-	-	-	-	-	-

Очевидно, перехід з стану a_5 до a_4 не дозволений, бо перший стан у підмножині b_2 не може бути досягненим якщо користувач вийшов з аканту. Заповнимо наші переходи у матрицю переходів станів.

Нехай множина впорядкованих пар переходу станів наступна

W = {(u; u) ; (u; a1) ; (a1; a1) ; (a1; a2) ; (a2; a2) ; (a2; a3) ; (a2; a4) ; (a2; a5) ; (a3; a2) ; (a3; a3) ; (a3; a4) ; (a3; a5) ; (a4; a2) ; (a4; a3) ; (a4; a4) ; (a4; a5) ; (a5; a1) }

Тоді можна подати представлення матриці W у двійкому вигляді:

Таблиця 2.2 – двійкова матриця переходу станів компонентів W

	u	a1	a2	a3	a4	a5
u	1*	1	0	0	0	0
a1	0	1	1	0	0	0
a2	0	0	1	1	1	1
a3	0	0	1	1	1	1
a4	0	0	1	1	1	1

a5	0	1	0	0	0	0

У випадку, коли стан недозволено, заповнюється та застосовується матриця S, що містить в собі набір безпечних компонентів.

Таблиця 2.3 – матриця переходу веб-сторінок у безпечні стани

	u	a1	a2	a3	a4	a5
u	NA	a1	a1	a1	a1	a1
a1	a5	a1	a1	a1	a1	a1
a2	a5	a5	a5	a1	a1	a5
a3	a5	a5	a5	a5	a5	a5
a4	a5	a5	a5	a5	a5	a5
a5	a5	a5	a5	a5	a5	a5

2.7 Висновок до розділу 2

У цьому розділі ми проаналізували концепцію типового використання "Білого списку" у сфері IT, а також розробив методику використання моделі "Білого списку" для розробки захищених веб-аплікацій. Також був проілюстрований приклад використання цієї моделі. Я вважаю, що можна зробити висновок, що будь-яку концепцію захисту та розмежування доступу можливо адаптувати під будь-який конкретний процес, а конкретно під процес розробки захищених веб-аплікацій. Згідно до аналізу створеного мною прикладу модель "Білого списку" являється працездатною та успішно забезпечує захищеність, як процесу розробки, так і кінцевого продукту.

3 ДОСЛІДЖЕННЯ ТЕХНОЛОГІЙ ДЛЯ СТВОРЕННЯ СИСТЕМИ БЕЗБЕКИ ВЕБ-ЗАСТОСУНКУ

В цьому розділі буде розглянуто сучасні технології, які можуть бути застосовані для побудови додатку з елементами соціальної мережі на мові C#.

Спочатку буде розглянуто можливості технології ASP.NET, а також буде надано опис технологій, які саме лежать в основі ASP.NET. Далі детальніше будуть описані такі шаблони, як: ASP.NET MVC, ASP.NET Web API та ASP.NET Core.

Найцікавішим є ASP.NET Core, адже даний фреймворк повністю переписаний, і об’єднує існувавші раніше окремі технологій ASP.NET MVC та ASP.NET Web API у єдиний програмний підхід. Також може виникати питання, для чого взагалі використовувати C#, так як існує досить багато альтернативних мов, такі як Java, C++ та Swift, що на сьогодні являються досить перспективними і цікавими мовами програмування.

Справді, проте мова C# має досить велику історію, багато корисних конструкцій, чималу кількість фреймворків, а також велику спільноту, яка по сьогодні підтримує технології .NET. Тому хотілося б розглянути основні та найбільш цікаві можливості даної мови:

– делегати - інкапсульовані сигнатури методів, які підтримують повідомлення про події з безпекою типів;

– властивості (Properties), що виступають в ролі методів доступу для публічно закритих змінних;

– атрибути що допомагають декларувати метадані типів під час виконання;

– вбудовані коментарі XML-документації;

– LINQ, що пропонує досить широкі можливості для створення запитів використовуючи різні джерела даних;

– підтримка асинхронних операцій;

– використання Task Parallel Library - бібліотеки для використання паралельного програмування в .NET;

– використання широкого спектру класів .NET та фреймворків.

Як показано виже, дана мова програмування має багато корисних можливостей. Вона досить схожа на C++ проте легша у вивченні.

3.1 Платформа ASP.NET

ASP.NET – платформа для розробки веб-застосунків та веб-сервісів, що була розроблена у компаніїMicrosoft. Дана платформа є основною частиною платформи Microsoft.NET та розвитком старішої технології Microsoft ASP. В даний момент ASP.NET Core 2.0.ASP.NET це остання версія даної платформи, зовні схожа на стару версію ASP , та дозволяє швидко мігрувати на ASP.NET. Але при всьому цьому внутрішні механізми ASP.NET суттєво відрізняється від ASP, вона має всі можливості, які надаються цією платформою, по причині заснування на платформі .NET.

Після представлення сервера Internet Information Services 4.0 в 1997 році, розпочалося дослідження можливості створення нової моделі веб-застосунку, створення якого понесе за собою зменшення скарг на ASP, які пов’язані з відділенням від змісту, що в свою чергу дозволить писати «чистий» код. Таку роботу було доручено Скотту Гутрі, що прийшов на роботу в Microsoft та Марку Андерсу, менеджеру команди IIS. Початковий проект було розроблено на протязі двох місяців, код первісного прототипу був написаний Гутрі під час різдвяних канікул. Назва початкового проекту «XSP»; «буква X при самому початку нічого не означала, як пояснив Гутрі, на той час всі технології починалися з X, тому й було обрану таку назву. XML починається з X, XSLT починається з X». Початковий прототип XSP було написано на Java, але з часом було прийнято рішення створити нову платформу на Common Language Runtime(CLR), оскільки у компанії Microsoft закінчувалась ліцензія на платформу Java, як пояснили у компанії це був великий ризик, оскільки CLR, як і XSP, перебувала у стадії розробки.

Корпорація Майкрософт рекомендує використовувати в динамічному коді програми "Code-behind model", яка розміщує цей код в окремому файлі або

спеціально позначеному тегу. Файли коду, як правило, мають імена "MyPage.aspx.cs" або "MyPage.aspx.vb", а файл сторінки — "MyPage.aspx" (таке ж саме ім'я, як і у файла сторінки (ASPX), але з розширенням, що визначає мову програмування).

Ця практика використовується в багатьох IDE в тому числі і в Visual Studio. Також, у форматі веб-додатків, "Pagename.aspx.cs" являється частковим класом, який зв'язаний з файлом "Pagename.designer.cs".

Файл дизайнера — це файл, який створюється з файлу ASPX автоматично, і дозволяє розробнику робити посилання на компоненти файлу ASPX з файлу CS без потреби оголошувати їх вручну, як це було в попередніх версіях ASP.NET. Використовуючи розглянутий стиль програмування, розробник пише код, який реагує на різні події, такі як взаємодія з елементом керування або завантаження сторінки, а не лише на процедурний перегляд документа. "Code-behind model" в ASP.NET відрізняється від класичного підходу ASP, оскільки він спонукає розробника створювати програмне забезпечення, відокремлюючи рівень відображення та логіку.

В теорії, це дозволить веб-дизайнерам зосередитись більше на розмітці дизайну веб-додатку, звертаючи менше уваги на виникнення помилок в програмному коді, який його запускає. Цей підхід схожий на відокремлення контролера від відображення в рамках підходу MVC (Model–View–Controller).

В .NET використовується технологія рендерингу "Відвіданих композитів" ("Visited composited"). Під час компіляції, файл шаблону переходить в код ініціалізації, який в свою чергу створює дерево упарвління (композит), що представляє собою вихідну модель. Літерний текст переходить в екземпляри класу "Literal control", а елементи керування - сервером трансформується в екземпляри конкретного класу керування. Код ініціалізації об'єднується з кодом, написаним розробником (зазвичай шляхом монтажу декількох часткових класів), і після цього переходить до потрібного класу сторінки.

Конкретні запити на веб-сторінку обробляються у декілька етапів. По-перше, створюється інстанс класу цієюї сторінки та виконується її код ініціалізації. Це створює початкове дерево управління, яке використовується методами сторінки у

наступних кроках. Оскільки кожен вузол у дереві є елементом управління, представлений як екземпляр класу, програмна логіка може змінювати структуру дерева, а також маніпулювати властивостями та методами окремих вузлів. Нарешті, на етапі візуалізації користувач веб-сторінки відвідує кожен вузол у дереві, і кожен вузол сам викликає методи відвідувача. Вихідний код веб-сторінки у вигляді HTML надсилається клієнту веб-сторінки.

Коли запит оброблено, екземпляр класу сторінки підлегле сбірці сміття, а разом з ним і все дерево управління. Це може викликати непорозуміння у програмістів-початківців ASP.NET, які взаємодіють з членів об'єкту класу, які відкидаються з кожним циклом запиту-відповіді сторінки.

Не дивлячись на те, що ASP.NET і бере свою назву від давньої технології Microsoft ASP, вона кардинально від неї відрізняється. Microsoft ASP.NET була подумбована як нова технологія, з іншою кодовою базою, будучи побудованою на Common Language Runtime (CLR). CLR є середовищем виконання для всіх застосунків Microsoft.NET. Розробники програмного забезпечення мають можливість писати програмний код для ASP.NET будь-якою мовою програмування, що входять у пакет .NET Framework (C#, Visual Basic.NET, JScript.NET ті інші). ASP.NET має кращі показники швидкості виконання в порівнянні зі скриптовими технологіями, які виконуються в браузері. Це відбувається завдяки специфіці CLR - при першому зверненні доділянки коду, останній компілюється і поміщається в спеціальний кеш, і згодом тільки виконується, не вимагаючи витрат часу на парсинг,оптимізацію, і так далі, на відміну від інтерпретованого коду скриптових технологій.

3.2 Технологія ASP.NET Web API

Технологія Web API створено за допомогою додавання в ASP.NET MVC фреймворк контролера спеціального виду. Даний вид контролерів під назвою API, має такі характеристики:

- замість об’єктів типу ActionResult, вони отримують об’єкти моделей;

– на основі HTTP-методу, що використовується в запиті, обираються методи.

На рисунку 3.1 зображено Структура додатку, написаного за допомогою технології ASP.NET Web API.

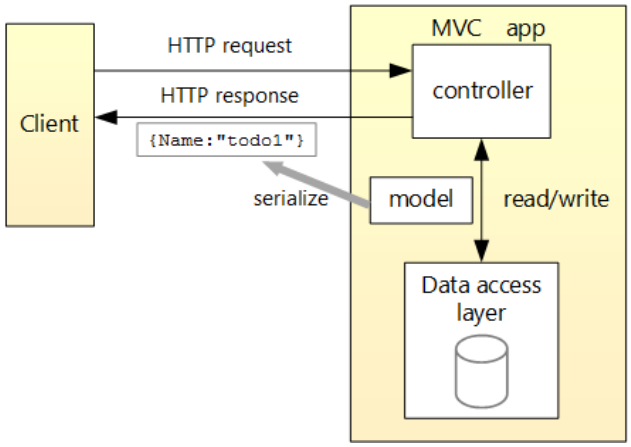

Рисунок 3.1 – Структура застосунку, розробленого з використанням фреймворку ASP.NET Web API

Моделі які отримуються за допомогою методу роботи контролера API, відправляються клієнту закодованими у форматі JSON. Контроллери Api використовуються для направлення веб-служб даних, по цій причині вони не мають підтримки уявлення, компонування та будь-яких інших засобів, що використовуються для генерації HTML-розмітки у створенні додатків.

У контролера API відсутня можливість генерування HTML-розмітки, це є причиною, по якій, односторінкові дадатки використовують комбіновані прийоми ASP.NET MVC фреймворк з WEB API. Для доставки HTML-вмісту, необхідно виконати кроки для доставки, які інфраструктура ASP.NET MVC фреймворк реалізує, включаючи авторизацію, аутентифікацію, вибір та візуалізацію уявлення. Після доставки HTML-вмісту до браузера, виконується обробка контролером Web API, за допомогою запитів Ajax.

При використанні звичайних контролерів, можна створювати та використовувати методи дій, що повертають дані JSON що підтримують Ajax, але

використання контролера API дозволяє використати альтернативний підхід. Даний підхід прогнозує відокремлення дій, що належать до даних, від дій, що відносяться до подання та швидко і просто створюють універсальні додатки WEB API.

3.3 ASP.NET Core. Актуальність використання відносно застарілих версій платформи ASP.NET

На рисунку 3.2 зображено структуру додатків, написаних за допомогою технології ASP.NET Core.

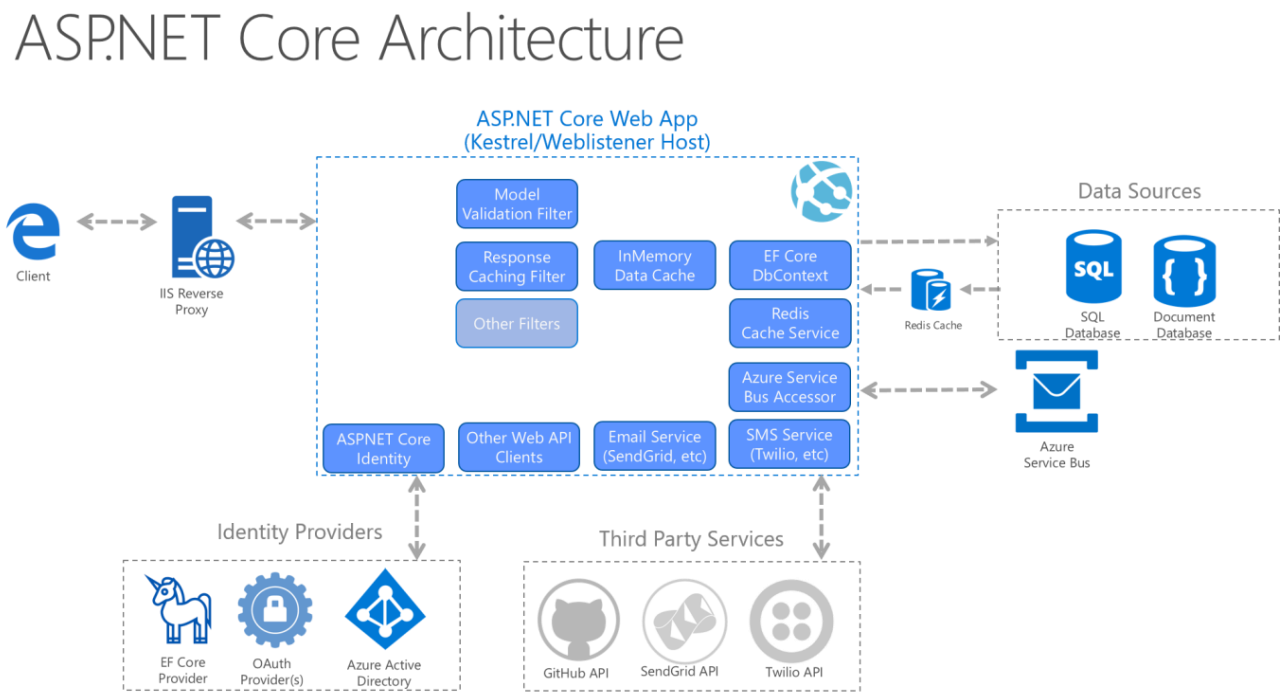

Рисунок 3.2 – Схема застосунків, розроблених з використанням технології ASP.NET Core

ASP.NET Core - це серверна технологія від компанії Microsoft, призначена для створення веб-додатків різних масштабів: від невеликих веб-сайтів до великих веб-порталів і веб-сервісів.

З одного боку, ASP.NET Core є новим витком розвитку платформи ASP.NET. Та з іншого боку, це не черговий реліз ASP.NET Framework. Вихід ASP.NET Core,

фактично, означає революцію всієї платформи, її якісна заміна. Розробка над платформою почалася ще в 2014 році. Тоді платформа мала кодову назву ASP.NET vNext. У червні 2016 року було випущено перший реліз платформи. В травні 2018 року світ побачила версія ASP.NET Core 2.1, яка власне і охоплена в поточному керівництві. ASP.NET Core є opensource-фреймворком, на відміну від ASP.NET Core. Всі вихідні файли фреймворку доступні на GitHub. Платформа ASP.NET Core базується на крос-платформенному середовищу .NET Core, яке може бути розгорнуте на операційних системах, які займають більшість і світі по використанню: Microsoft,Windows, Mac OS X, Linux.

Таким чином, ASP.NET Core дає можливість створювати незалежні від плафторми додатки. Windows, як середовище для розробки і розгортання програмного забезпечення досі превалює для користувачів .NET, але тепер ми не обмежені тільки цією операційною системою.

Тобто, ми можемо утримувати веб-додатки не тільки на операційній системі Windows, але і на Linux і Mac OS. Для розгортання веб-додатків можна використовувати традиційний IIS (Internet Information Services) на Windows, або крос-платформний веб-сервер Kestrel. Хоча платформа ASP.NET Core і націлена переважно на використання .NET Core, але технологія також може працювати і з повною версією фреймворка .NET. Завдяки модульності фреймворка всі необхідні компоненти веб-додатків можуть підгружатися як окремі модулі через Nuget пакетний менеджер. Крім того, на відміну від попередніх версій платформи немає вимоги використовувати стару бібліотеку System.Web.dll.

ASP.NET Core включає фреймворк MVC, який об'єднує функціональність популярних ферймворків MVC, Web API і Web Pages. У попередніх версіях платформи NET, ці технології реалізувалися окремо, і мали багато функціональності, яка дублювалася. На сьогодні, технології MVC і Web API об'єднали в одну програмну модель під назвою ASP.NET Core MVC, а технологія Web Forms - повністю залишилася в минулому. Крім об'єднання розглянутих технологій в одну модель MVC, також було доданий ряд нових функцій. Однією з таких функцій є тег-хелпери (tag helper), які дозволяють більш зручно поєднувати код, написаний на C# з

синтаксисом html.

ASP.NET Core характеризується гарною гнучкістю та розширюваністю. Даний фреймворк побудований з набору незалежних компонентів, і ми можемо або використовувати вбудовану реалізацію існуючих компонентів, розширити їх за допомогою механізму унаслідування, або створити зовсім нові компоненти і застосовувати їх зі своїм функціоналом.

Також було спрощено механізм керування залежностями та конфігурація проекту. Фреймворк тепер має свій простий та легкий контейнер для вбудовування залежностей, і тому більше немає потреби використовувати сторонні бібліотеки, такі як Autofac або Ninject. Проте при бажанні їх також можна використовувати.

В якості середовища для розробки ми можемо використовувати IDE MS Visual Studio, починаючи з версії Visual Studio 2015. Крім того, ми також можемо створювати програмне забезпечення в середовищі MS Visual Studio Code, яке є крос-платформним рішенням і може працювати на різних операційних системах, як на Windows, так і на Mac OS X і Linux. Для обробки запитів використовується новий конвеєр HTTP, який заснований на специфікації OWIN і компонентах Katana. Модульність нового конвеєру HTTP запитів дозволяє легко додавати свої власні необхідні компоненти.

3.4 Методи структуризації коду при написанні ASP.NET Core

Кожен великий проект використовує паттерни для кращої структуризації коду та його підтримки в майбутньому. Паттерн – це “рекомендація” розробникам для написання якісного коду, тобто це не панацея і будь-який розробник в праві відмовитися від використання паттернів.

Розглянемо основні із них, що використовуються при написанні програм з використанням технології ASP.NET Core:

– використання багатошарової архітектури, розділення на шари (n-tier architecture);

– використання паттерна MVC(модель, представлення, контроллер), даний паттерн використовується ще й у веб-додатках для розшарування логіки, графічного інтерфейсу та доступу до даних;

– вставка залежностей, використання шаблону dependency injection (DI), класи не повинні залежати від конкретної реалізації,а залежати від інтерфейсів;

– репозиторій (repository) – даний патерн застосовується як засіб доступу до сховища бази даних;

– використання принципів SOLID.

На рисунку 3.3 зображено приклад структури багаторівневої архітектури.

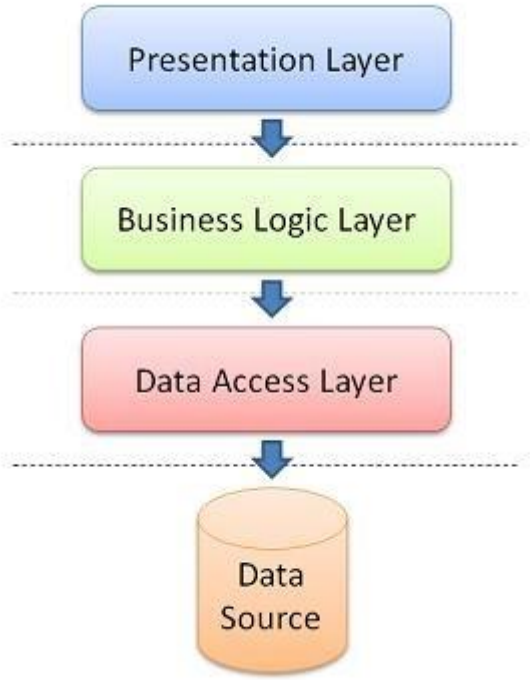

Рисунок 3.3 – Приклад багаторівневої архітектури

Багаторівнева архітектура[9] – вид архітектури додатків, в якій розділяються функції представлення, обробки і зберігання даних на рівні. Найбільш поширеним різновидом багаторівневої архітектури є трирівнева архітектура.

N-рівнева архітектура додатка надає модель, за допомогою якої розробники програмного забезпечення можуть створювати гнучкі і повторно-використовувані частини програм. Поділяючи додаток на рівні абстракції, розробники мають перевагу

гнучкого внесення змін в якийсь певний шар за необхідності, замість того, щоб переробляти додаток цілком. Трирівнева архітектура традиційно складається з рівнів представлення, бізнес логіки і доступу до даних.

Не дивлячись на те, що поняття шару і рівня найчастіше використовуються як тотожні, багато сходяться на думці, що відмінність між ними є. Різниця в тому, що шар - це механізм логічного структурування компонентів, з яких складається програмне рішення, в той час як рівень - це механізм фізичного структурування інфраструктури системи. Трирівневе рішення може бути легко розгорнуто на єдиному рівні, такому як персональна робоча станція.

Цей шаблон передбачає поділ системи на три взаємопов'язані частини: модель даних, вигляд (інтерфейс користувача) та модуль управління. Він використовується для відокремлення моделі (шару даних) від інтерфейсу користувача (шару представлення) так, щоб зміни інтерфейсу користувача мінімально впливали на роботу з даними, а зміни в моделі могли здійснюватися без змін інтерфейсу користувача.

Мета шаблону — гнучкий дизайн програмного забезпечення, який повинен полегшувати подальші зміни чи розширення програм, а також надавати можливість повторного використання окремих компонентів програми. Крім того використання цього шаблону у великих системах сприяє впорядкованості їхньої структури і робить їх більш зрозумілими за рахунок зменшення складності.

Інжекція залежностей - це метод, за допомогою якого один об'єкт (або статичний метод) постачає залежності іншого об'єкта. Залежність - це об'єкт, який можна використовувати (служба). Ін'єкція - це передача залежності на залежний об'єкт (клієнт), який буде використовувати його. Послуга входить до складу держави клієнта. Передача послуги клієнту, а не дозвіл клієнту побудувати або знайти послугу, є основною вимогою шаблону.

Метою ін'єкції залежностей є досягнення розділення проблем будівництва та використання об'єктів.

Ін'єкція залежності є однією з форм більш широкої техніки інверсії контролю. Як і в інших формах інверсії контролю, інжекція залежностей підтримує принцип

інверсії залежностей. Клієнт делегує відповідальність за надання своїх залежностей зовнішньому коду (інжектору). Клієнту не дозволяється викликати інжекторний код, це ін'єкційний код, який конструює служби і викликає клієнта, щоб ввести їх. Це означає, що клієнтський код не повинен знати про код ін'єкції, про те, як побудувати послуги або навіть які фактичні послуги він використовує; клієнт повинен знати лише про внутрішні інтерфейси послуг, оскільки вони визначають, як клієнт може користуватися послугами. Це розділяє обов'язки використання та будівництва.

Для прийняття ін’єкції залежностей (DI) існує три загальні способи: ін’єкція засновника, конструктора та інтерфейсу. Інжектори сетера та конструктора різняться тим, що вони використовуються у різних ситуаціях. Інжекція інтерфейсу надає можливість контролю власної ін’єкції, це є головною відмінністю даної ін’єкції. Кожен з способів наполягає, щоб кожен з інжекторів відповідав за введення клієнта та його залежність один від одного.

На рисунку 3.4 зображено схему паттерна Repository.

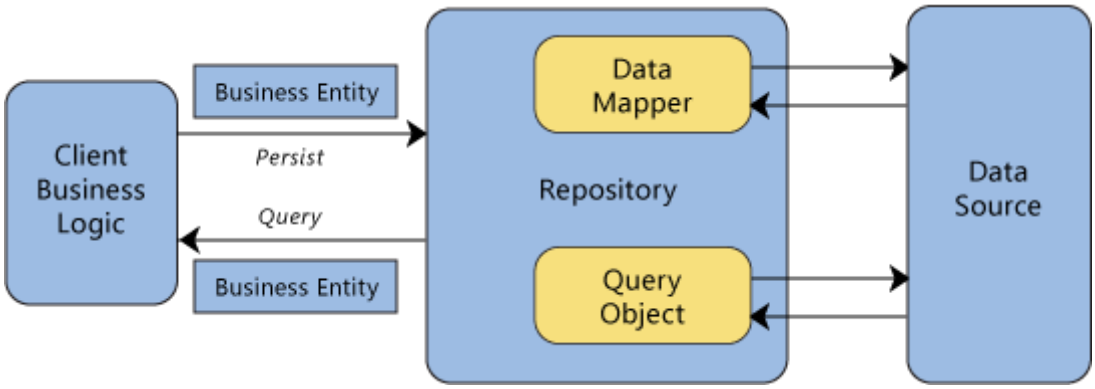

Рисунок 3.4 – Призначення паттерна Repository

Між рівнями області визначення та розподілу (domain anddata mapping layers), з використанням інтерфейсу, який є схожим з елементами для доступу об’єктів визначення. Складну систему можна спростити за допомогою рівня що додається, наприклад за допомогою Data Mapper, який би реалізовував ізоляцію від бази даних до доку доступу. В дані системи можна додати ще один шар поверх шару абстракції даних, у якому б можна було збирати код для створення засобів.

Дані підходи стають ще більш важливими у області визначення безлічі класів та при запитах, що мають складну структуру. При таких випадках, додавання даного рівня дозволяє значно скоротити повторювання коду запитів.

Шаблон Repository є обгорткою над джерелом даних, працюючи, як звичайна колекція об'єктів області визначення. Клієнти об'єкта- сховища (Repository) створюють опис запиту декларативно і направляють їх до реального сховища для обробки. Об'єкти можуть бути додані або видалені з сховища, як ніби вони формують просту колекцію об'єктів. А код розподілу даних, інкапсульований в об'єкті Repository, подбає про підключення до джерела даних та маніпуляцію над даними при відповідних операціях непомітно для розробника. У двох словах, шаблон Repository інкапсулює об'єкти, представлені в сховищі даних і операції, що виконуються над ними, надаючи об'єктно-орієнтоване уявлення реальних даних. Repository використовується для досягнення повного поділу і односторонньої залежності між рівнями області визначення і розподілу даних.

SOLID – це абревіатура складена з перших літер п’яти базових принципів розробки підтримуваного, розширюваного дизайну об’єктно-орієнтованого програмування, запропонована Робертом Мартіном. Принципи SOLID використовуються для розробки таких програмних систем, які, з великою ймовірністю, будуть тривалий час розвиватися, розширятися і підтримуватися.

Принцип єдиної відповідальності (Single Responsibility Principle) можна сформулювати так: окрема одиниця програмного забезпечення має містити тільки одну причину для зміни. Під відповідальністю тут розуміється набір функцій одниці програмного забезпечення, які виконують єдине завдання. Суть принципу єдиної відповідальності полягає в тому, що одиниця програмного забезпечення має виконувати одну неподільну задачу. Весь функціонал одиниці повинен бути цілісним, володіти високою зв'язністю (high cohesion). Конкретна реалізація принципу залежить від контексту в якому він використовується. В даному випадку важливо розуміти, як змінюється одиниця. Якщо вона виконує кілька різних функцій, і вони змінюються окремо, то це якраз той випадок, коли порушується цей принцип і має сенс розділити обов’язки між різними одиницями. Тобто, одиниця програмного забезпечення має

кулька причин для зміни.

Принцип відкритості/закритості (Open/Closed Principle) можна сформулювати наступним чином: об’єкти програми повинні бути відкриті для розширення, але закриті для змін. Суть даного підходу полягає в тому, що система повинна бути побудована так, що всі її подальші зміни повинні бути реалізовані не за допомогою зміни вже існуючого коду, а додавання нового.

Принцип підстановки Лісков (Liskov Substitution Principle) являє собою правило для створення ієрархій наслідування. Його можна сформувати наступним чином: повинна бути можливість підставити замість базового типу будь-який його дочірній тип, при цьому не зламавши вже існуючу імплементацію. Даний принцип допомагає чіткіше сформулювати ієрархію класів, визначити функціональність для базових і дочірніх класів і як наслідок уникнути можливих проблем при застосуванні поліморфізму.

Принцип поділу інтерфейсів (Interface Segregation Principle) відноситься до тих випадків, коли інтерфейси мають, дуже роздуту кількість методів, не всі методи і властивості якого використовуються і можуть бути необхідні користувачеві. Таким чином, реалізації інтерфейсів вийдуть надмірними або "жирними". Принцип поділу інтерфейсів можна сформулювати так: клієнти не повинні бути вимушені залежати від методів, які їм не потрібні. При порушенні цього принципу клієнт, який використовує певний інтерфейс з усіма його методами, залежить від необхідності імплементувати методи, якими не користується, і тому є чутливим до змін в цих методах.

У результаті ми приходимо до жорсткої залежності (high coupling) між різними частинами інтерфейсу, які можуть бути не пов'язані при його реалізації. В цьому випадку інтерфейс класу поділяється на окремі частини, які становлять окремі інтерфейси. Потім ці інтерфейси незалежно один від одного можуть застосовуватися і змінюватися. В результаті застосування принципу поділу інтерфейсів робить систему слабозв’язанною (low coupled), і тим самим її легше змінювати і оновлювати.

Принцип інверсії залежностей (Dependency Inversion Principle) служить для створення слабозв’язанних сутностей, які легко модифікувати, оновлювати. Цей

принцип можна сформулювати таким чином: модулі верхнього рівня не повинні залежати від модулів нижнього рівня. І ті й інші повинні залежати від абстракцій. Абстракції не повинні залежати від деталей. Деталі повинні залежати від абстракцій.

3.5 Фреймворк для роботи з базою даних Entity Framework Core

На рисунку 3.5 зображенно структуру роботи технології EF Core.

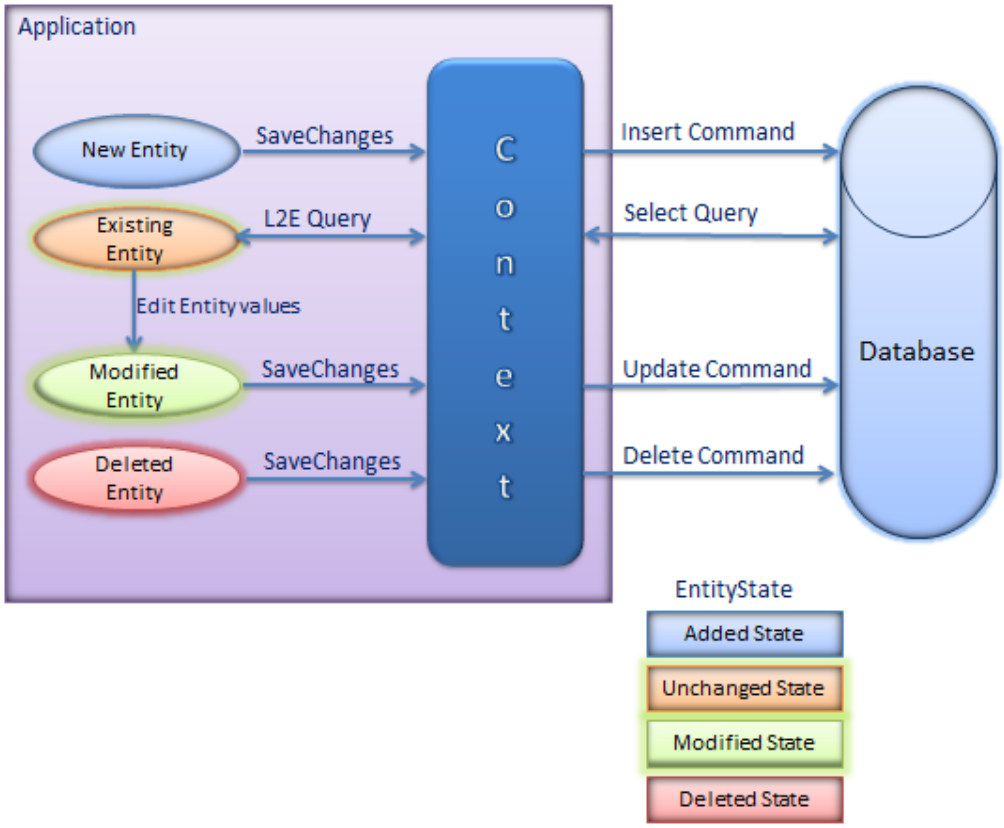

Рисунок 3.5 – Структура роботи технології EF Core

Entity Framework Core (EF Core) це технологя від компанії Microsoft, яка являє собою об'єктно-орієнтовану, легку і зручну для розширення технологію для доступу до даних. EF Core є ORM-інструментом (object-relational mapping - відображення даних на реальних об'єктах). EF Core дозволяє працювати з базами даних, і водночас має більш високий рівень абстракції: ця технологія дозволяє абстрагуватися від самої бази даних і її таблиць і працювати з даними у вигляді об'єктів незалежно від типу

сховища. Якщо на фізичному рівні ми оперуємо таблицями, індексами, ключами, то на рівні програми, ми вже працюємо з об'єктами, які нам пропонує та будує Entity Framework.

Entity Framework Core підтримує безліч різних видів баз даних. Таким чином, ми можемо працювати з будь-якої СУБД використовуючи EF Core, якщо для неї є необхідний провайдер.

На сьогодні компанія Microsoft надає ряд вбудованих провайдерів для роботи з наступними базами даних: MS SQL Server, SQLite, PostgreSQL. Також існують провайдери від сторонніх постачальників, наприклад, для MySQL. Також варто відзначити, що EF Core надає універсальний API для роботи з даними. І якщо потрібно змінити базу даних, що використовується, то основні зміни в програмі будуть стосуватися конфігурації і налаштування підключення до відповідного провайдера даних. А код, який безпосередньо працює з даними, залишиться без змін.

Entity Framework Core багато успадкував від своїх попередників, зокрема, Entity Framework 6. У той же час треба розуміти, що EF Core - це не нова версія по відношенню до EF 6, а зовсім інша технологія, хоча в цілому принципи роботи у них будуть збігатися. Тому в рамках EF Core використовується своя система версій. Поточна версія - 2.0 була випущена в серпні 2017 року. І технологія продовжує розвиватися. Що вже є в EF Core, а що тільки планується додати, можна подивитися в роудмапе на гітхабе.

Як технологія доступу до даних Entity Framework Core може використовуватися на різних платформах стека .NET. Це і стандартні платформи типу Windows Forms, консольні додатки, WPF, ASP.NET 4.6 / 4.7. Це і нові технології як UWP і ASP.NET Core. При цьому кроссплатформенная природа EF Core дозволяє задіяти її не тільки на ОС Windows, але і на Linux і Mac OS X.

Центральної концепцією Entity Framework є поняття сутності або entity. Сутність визначає набір даних, які пов'язані з певним об'єктом. Тому дана технологія передбачає роботу не з таблицями, а з об'єктами і їх колекціями.

Будь-яка сутність, як і будь-який об'єкт з реального світу, має низку

властивостей. Наприклад, якщо сутність описує людини, то ми можемо виділити такі властивості, як ім'я, прізвище, зріст, вік. Властивості необов'язково представляють прості дані типу int або string, але можуть також представляти і більш комплексні типи даних. І у кожної сутності може бути одна або кілька властивостей, які будуть відрізняти цю сутність від інших і будуть унікально визначати цю сутність. Подібні властивості називають ключами.

При цьому суті можуть бути пов'язані асоціативної зв'язком один-ко-многим, один-ко-одному і багато-до-багатьох, подібно до того, як в реальній базі даних відбувається зв'язок через зовнішні ключі.

Відмінною рисою Entity Framework Core, як технології ORM, є використання запитів LINQ для вибірки даних з БД. За допомогою LINQ ми можемо створювати різні запити на вибірку об'єктів, в тому числі пов'язаних різними асоціативними зв'язками. А Entity Framework при виконання запиту транслює вираження LINQ в вирази, зрозумілі для конкретної СУБД (як правило, в вирази SQL).

3.6 Технологія для роботи з базою даних ADO.NET

На рисунку 3.6 зображено структуру технології ADO.NET.

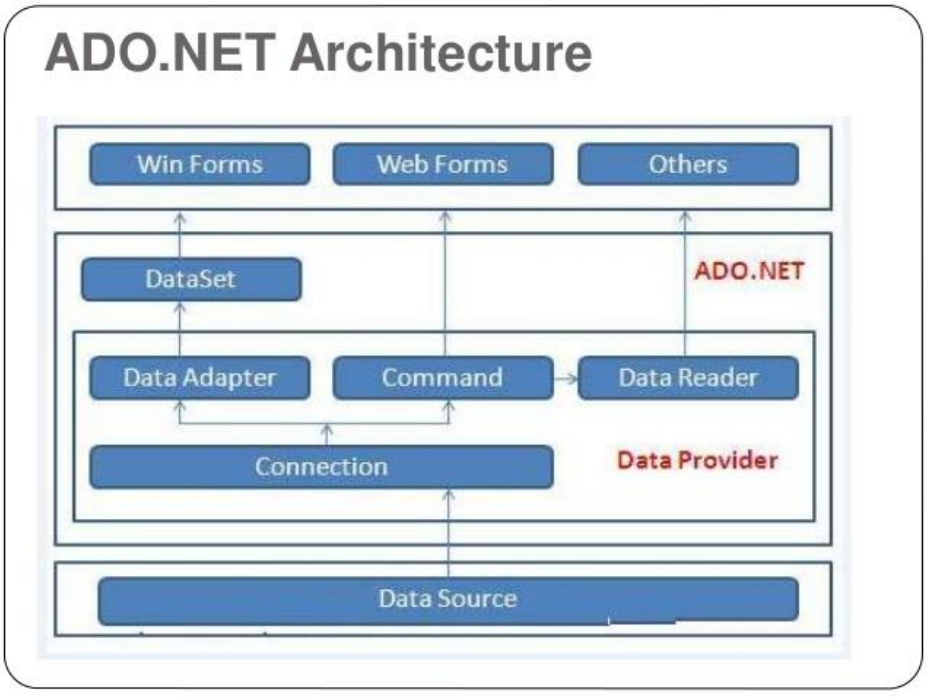

Рисунок 3.6 – Структура технології ADO.NET

На даний момент велике значення має робота з даними. Для зберігання даних існує безліч різних систем управління базами даних: MS SQL Server, Oracle,MySQL та багато інших.

Більшість додатків які мають велику структуру, у своїй більшості використовують, для зберігання даних, ці системи управління базами даних. Але для того щоб побудувати зв’язок між базою даних та додатком побудованим на C#, необхідне використання посередника. А таким посередником виступає технологія ADO.NET.ADO.NET, яка надає технологію, що базується на платформі .NET Framework.

Дана технологія представляє собою набір класів, за допомогою яких можна відправляти запити до баз даних, отримувати відповідь, встановлювати підключення до бази даних і виконувати ряд інших операцій. Але важливо звернути увагу на те, що системи управління баз даних може бути чимало. По своїй реалізації вони можуть відрізнятись. MS SQL для створення запитів використовує мову T-SQL, а в свою чергу MySQL та Oracle використовують мову PL-SQL. Кожна база даних може мати свої різні типи даних, а також можуть відрізнятися в любих інших характеристиках. Функціонал ADO.NET спроектований для того, щоб розробники мали змогу використання уніфікованого інтерфейсу для використання різних систем управління базами даних.

Основний інтерфейс взаємодії з даними бази в ADO.NET представляється обмеженим колом об'єктів: Connection, Command, DataReader, DataSet і DataAdapter. Об'єкт Connection допомагає налаштувати підключення до джерела даних. Об'єкт Command дозволяє виконувати різного роду операції над даними в БД. Об'єкт DataReader зчитує дані, які були отримані в результаті запиту до БД. Об'єкт DataSet дозволяє працювати з даними незалежно від типу БД, а також призначений для зберігання цих даних в БД. Об'єкт DataAdapter в свою чергу є посередником між DataSet і джерелом даних.

Через розглянуті об'єкти і буде йти взаємодія з даними в БД. Проте щоб використовувати однаковий набір об'єктів для різних видів БД, необхідний відповідний провайдер для відповідної БД. Саме через провайдер даних в ADO.NET здійснюється взаємодія між програмою і базою даних. При цьому в ADO.NET може бути власний провайдер для кожного з джерел даних, який власне і визначає відповідну реалізацію вище розглянутих класів.

В ADO.NET за замовчуванням є наступні вбудовані провайдери:

– MS SQL Server;

– Oracle;

– OLE DB (Надає доступ до старих версій MS SQLServer, а також до БД Access, DB2, MySQL і Oracle);

– ODBC (провайдер для джерел, для яких немає своїх власних провайдерів даних);

– EntityClient - провайдер даних для технології ORM EntityFramework;

– для сервера SQL Server Compact 4.0.

Крім вище приведених провайдерів, які є вбудованими, існує також безліч інших варіантів, які використовуються для різних баз даних, наприклад, для MySQL.

ADO.NET по функціональності можна розбити на декілька рівнів: підключений та відключений. Кожен з провайдерів даних .NET використовує свою реалізацію варіантів об’єктів Connection, Command, DataReader, DataAdapter та інші, що складають собою підключений рівень. Тобто вони складають собою підключення до бази даних та взаємодіють з нею.

Зазвичай, реалізація даних об’єктів для окремих провайдерів у своїй назві мають префікс, що вказує на сам провайдер. Класи, такі як: DataSet, DataTable, DataRow, DataColumn та інші, організовують відключений рівень, оскільки після отримання інформації в DataSet є можливість працювати з нею незалежно від статусу підключення, тобто після отримання даних з бази даних можна відключитися від джерела інформації.

3.7 Цифровий підпис

За допомогою особистого ключа накладається електронний цифровий підпис, а за допомогою відкритого ключа перевіряється його валідність. Електронний цифровий підпис прирівнюється до власноручного підпису (печатки) за правовим статусом. Електронний підпис не визнається недійсним лише через те, що він має електронну форму або не має за основу посилений сертифікат ключа.

Підробка цифрового ключа неможлива за умови правильного зберігання власником особистого секретного ключа. Також не можливо підробити і електронний документ: не санкціоновано внесені будь-які зміни в текст документа, миттєво будуть виявлені.

Створюється особистий ключ ЕЦП на підставі абсолютно випадкових чисел, що створюються генератором випадкових чисел, а відкритий ключ обчислюється використовуючи секретний ключ ЕЦП так, щоб одержати другий з першого було неможливо. Особистий ключ ЕЦП являється послідовністю символів довжиною в 264 біта, яка призначена для створення Електронного цифрового підпису в електронних документах.

Працює особистий ключ тільки в парі з відкритим ключем. Особистий ключ необхідно зберігати в таємниці, адже якщо хтось його дізнається, то зможе підробити Електронний цифровий підпис власника. Документ для захищеності, повинен бути підписаний ЕЦП за допомогою особистого ключа, який повинен існувати тільки в одному екземплярі і тільки у його власника. Цьому особистому ключу у відповідності існує відкритий ключ, за допомогою якого можна перевірити відповідність підпису його власнику.

Щоб перевірити ЕЦП, для документів, що отримуються, використовується відкритий ключ. Використання відкритого ключа є можливим тільки з особистим ключем, з яким він використовується у парі. Відкритий ключ міститься в Сертифікаті відкритого ключа та підтверджує належність відкритого ключа ЕЦП певній особі. Окрім відкритого ключа, сертифікат відкритого ключа, також має в собі приватну інформацію про його власника, унікальний реєстраційний номер, термін дії

сертифікату відкритого ключа.

Для того щоб забезпечити цілісність даних, що представлені в сертифікаті, він підписується особистим ключем, за допомогою центру сертифікації ключів. Інформація про сертифікацію відкритого ключа, може бути опублікована на сайті відповідного центру сертифікації ключів, за договором про надання послуг ЕПЦ.

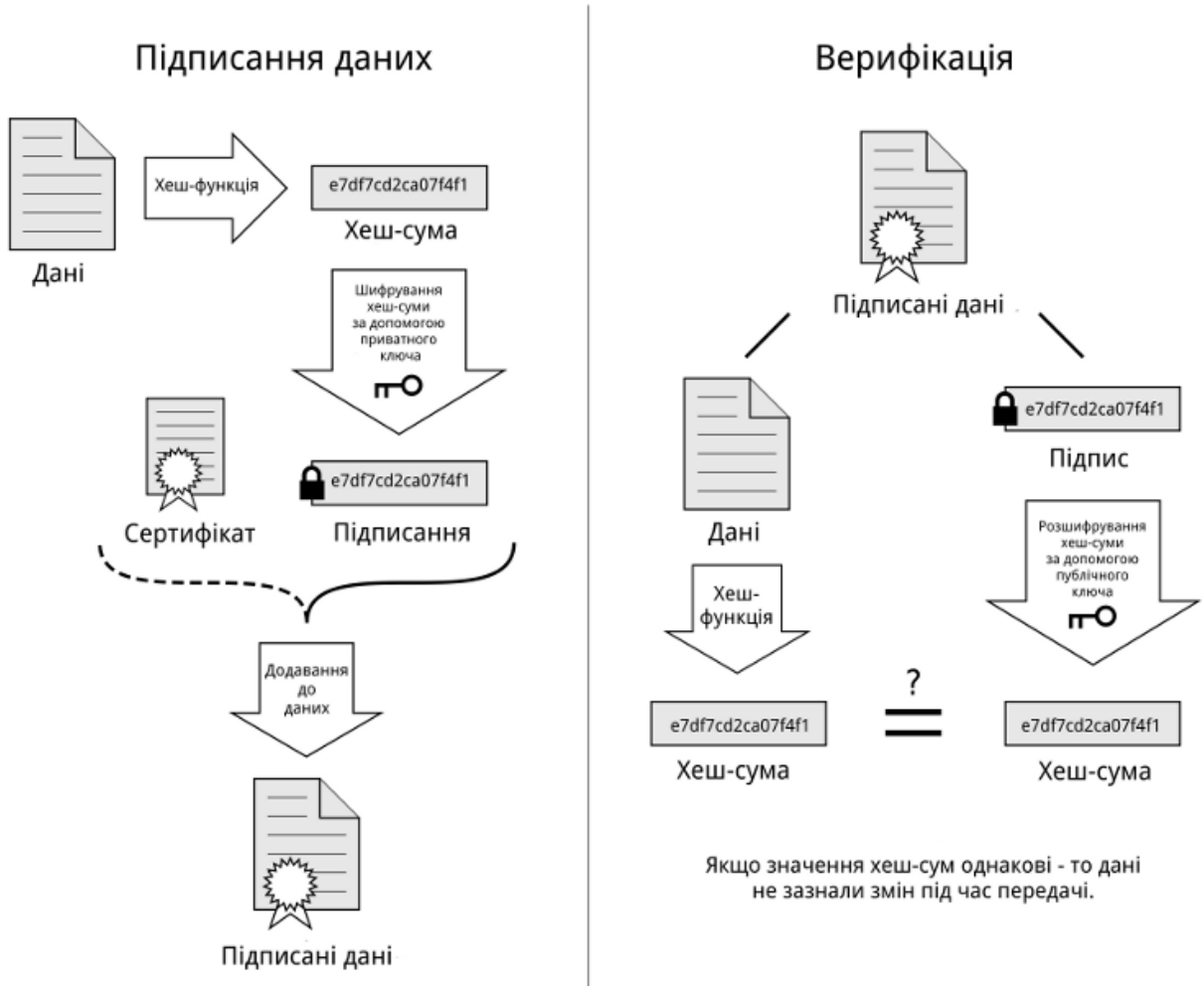

Рисунок 3.7 – Структура цифрового підпису даних

При підписанні електронного документа його початковий зміст не змінюється, а додається блок даних, так званий «Електронний цифровий підпис». Отримання цього блоку можна розділити на два етапи.

На першому етапі за допомогою програмного забезпечення і спеціальної математичної функції обчислюється так званий «відбиток повідомлення»

Цей відбиток має такі властивості:

- фіксовану довжину, незалежно від довжини повідомлення;
- унікальність відбитку для кожного повідомлення;
- неможливість відновлення повідомлення за його відбитком.

Таким чином, якщо документ був модифікований, то зміниться і його відбиток, що відобразиться при перевірці Електронного цифрового підпису.

На другому етапі відбиток документа шифрується за допомогою програмного забезпечення і особистого ключа автора.

Розшифрувати ЕЦП і одержати початковий відбиток, який відповідатиме документу, можна тільки використовуючи Сертифікат відкритого ключа автора.

Таким чином, обчислення відбитку документа захищає його від модифікації сторонніми особами після підписання, а шифрування особистим ключем автора підтверджує авторство документа.

Перевірка ЕЦП для отриманого документу, виконується в декілька етапів, на першому етапі адресат з використанням програмного забезпечення сертифікатом відкритого ключа автора, виконує розшифрування підписаного відбитку та отримує початковий відбиток документу.

З використанням програмного забезпечення та математичної функції, із документа, що було отримано, вираховується його відбиток. При виконанні перевірки ЕЦП виконується порівняння відбитків початкового та одержаного документів. Результатом перевірки є відповіль «вірно/невірно»

3.8 Шифрування Advanced Encryption Standard

Advanced Encryption Standard (AES) або Rijndael – алгоритм симетричного блочного шифрування (має блоковий розмір 128 біт, має ключ 128/192/256 біт), є стандартом шифрування, який прийнятий урядом США у результаті входу у фінал конкурсу AES. Алгоритм AES та алгоритм Реймена і Дейцмена, не являються одним і тим же. Алгоритм Рейндол має підтримку широкого діапазону розмірів блоку та

ключаю AES в свою чергу має фіксовану довжину у 128 біт, а у розмірі ключа можуть бути значення 128, 192 чи 256 біт. А Рейндол має підтримку блоку та ключа з кроком 32біт з діапазоном 128-256. По причині того, що фіксований розмір блоку AES працює з масивом 4 на 4 байти, це називається станом, якщо алгоритм має більший розмір блоку, то він має додаткові колонки. Наприклад, з ключем 128 біт, алгоритм має 10 раундів які послідовно виконують операції subBytes()–shiftRows()–mixcolumns() (у 10-му раунді пропускається)–xorRoundKey().

Функцією метода SubBytes() є незалежне оброблення байтів стану. Ця процедура за допомогою таблиці замін (S-box) здійснює нелінійну заміну байтів та забезпечує нелінійність алгоритму шифрування.

Метод ShiftRows обробляє рядки таблиці State таким чином: він зсуває рядок на r байтів по горизонталі, в залежності від номера рядка. Наприклад, цей метод присвоює нульовому рядку нуль (r = 0), першому рядку - один (r = 1) і так далі. Отже, після виклику метода ShiftRows кожна результуюча колонка буде складатися з байтів кожної початкової колонки. До речі, алгоритм Rijndael використовує механізм зсуву рядків, який є однаковим, як для 128-бітних, так і для 192-бітних рядків. Але, варто відзначити, що для алгоритм для блоків розміром 256 біт трохи інший, насамперед, відрізняється від інших тим, що для 2, 3, і 4 рядків відбувається зміщення на 1, 3, і 4 байти, відповідно. Простіше кажучи, цей процес можна описати, як перестановка байтів таблиці State.

У методі MixColumns реалізоване змішення чотирьох байтів кожної колонки таблиці State. В ньому використовується так звану зворотно-лінійну трансформацію. У методах AddRoundKey, RoundKey кожного раунду зливається зі State. Для кожного раунду Roundkey виходить з CipherKey, викликаючи метод KeyExpansion. Варто зауважити, що розмір кожного RoundKey є таким як і у State.

Функцією метода KeyExpansion є здійснення побітового XOR кожного байта State з кожним байтом RoundKey. Простіше кажучи, виконується так звана побайтова "виключна диз'юнкція" байт-ключа з байтами таблиці State.

3.9 Сервер автентифікації Microsoft Identity

Microsoft Identity Web - це набір бібліотеки ASP.NET Core, розширення Додавання підтримки перевірки підлинності та авторизації веб-додатків та веб-API, інтегрованих з платформою Microsoft Identity. Він надає зручний високопродуктивний API рівня, зв'язуючий ASP.NET Core, за допомогою промежуточного слова для перевірки підлинності та бібліотеки перевірки підлинності Майкрософт (MSAL) для .NET.

Це рішення, яке допомагає організаціям обмежувати привілейований доступ в існуючій і ізольованою Active Directory середовищі.Управління привілейованим доступом переслідує дві мети:

– відновлення контролю над скомпрометованої середовищем Active Directory за рахунок змісту окремої середовища бастіону, що не постраждала від шкідливих атак.

– ізольоване використання привілейованих облікових записів для зменшення ризику крадіжки облікових даних.

Надає наступні переваги:

– ізоляція або обмеження привілеїв. Його користувачі не мають привілеїв щодо облікових записів, які також використовуються для виконання завдань, які потребують привілейованого доступу, таких як перевірка електронної пошти або пошук в Інтернеті. Користувачі повинні запитувати привілеї. Запити затверджуються або відхиляються на основі політик MIM, визначених адміністратором PAM. Поки запит не затверджений, привілейований доступ не надається.

– підвищення рівня і додатковий захист. Ці нові заходи щодо вдосконалення перевірки автентичності та авторизації допомагають керувати життєвим циклом окремих облікових записів адміністраторів. Користувач може запросити підвищення прав облікового запису адміністратора, після чого цей запит проходить процедури MIM.

– більш докладний ведення журналу. Крім вбудованих робочих процесів MIM передбачені додаткові записи в журнал для PIM, за якими можна бачити запит, його авторизацію і події, що відбуваються після затвердження запиту.

– налаштовується процес. Робочі процеси MIM можна налаштувати для різних сценаріїв. Можна застосовувати кілька робочих процесів на основі параметрів запитувача користувача або запитаних ролей.

3.10 Запобігання SQL-ін’єкцій

SQL-ін’єкція - це механізм виконання будь-якого запиту до бази даних програми за допомогою параметра URL або поля форми. У разі використання стандартної мови Transact SQL існує можливість вставити шкідливий код в запит до бази. Внаслідок чого можуть бути отримані, змінені або видалені дані з таблиць, що існують в БД. Щоб запобігти цьому, потрібно використовувати параметризовані запити, які мають підтримку в більшості мов програмування, що використовуються при розробці веб-додатків.

Розглянемо запит: SELECT * FROM tableName WHERE column = 'parameterName'. Якщо зловмисник змінить значення “parameterName” на “OR '1'='1’”, то такий запит набере вигляду: SELECT * FROM table WHERE column = '' OR '1'='1’”. Оскільки результатом даного виразу '1' = '1' завжди буде “true”, зловмисник отримає доступ до даних з усіх таблиць бази даних. Цей прийом дозволяє виконати будь який довільний запит до БД, додавши в кінець даний вираз SQL. за допомогою параметризації можна легко усунути вразливість цього запиту. Наприклад, для програми, що написана з використанням PHP і MySQL, він має наступний вигляд:

```
$stmt = $pdo ->prepare ('SELECT * FROM tableName WHERE columnName = :value');
$stmt->execute (array ('value' => $parameterName));
```

3.11 Висновок до розділу 3

В даному розділі було оглянуто основні можливості фреймворку ASP.NET для побудови веб-додатків, а також найбільш поширені фреймворки для роботи з базами

даних Entity Framework Core та ADO.NET. Також було оглянуто різні версії ASP.NET та їх можливості, такі як ASP.NET Web API, ASP.NET MVC та ASP.NET Core. Окрім можливостей вищезгаданих фреймворків, було також розглянуто історії створення даних технологій, їх розвиток та призначення.

4 РОЗРОБКА ВЛАСНОГО РІШЕННЯ

Базуючись на інформації отриманій в попередніх розділах було вирішено розробляти систему захисту інформації веб-застосунків на базі вказаних технологій.

4.1 Принцип функціонування та побудування веб – серверів

Веб-сайти розміщуються на веб-серверах. Веб-сервер – це обчислювальна машина, на якій працює певна операційна система, та має можливість зберігати файли, наприклад:стилі CSS, HTML-документи, скриптові файли, зображення. Та реалізує доставку файлів на пристрій кінцевого користувача.

Також веб-сервери можуть взаємодіяти з базами даних, якщо вони були спроектовані для отримання інформації з баз даних та віддавати їх у певному форматі HTML[10].

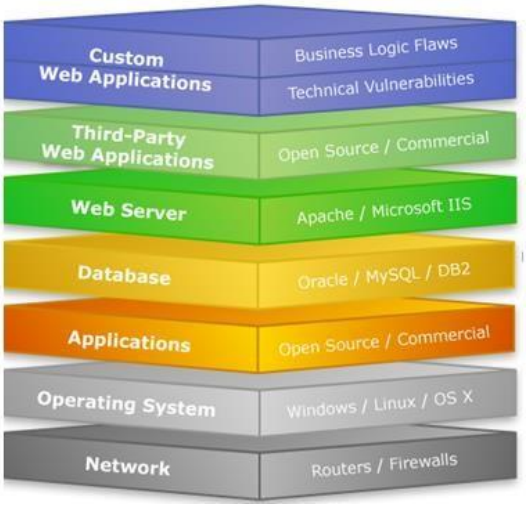

Рисунок 4.1 – Стек вразливостей веб-сервера

Якщо дивитися з точки зору програмного забезпечення, веб-сервер має в собі певну кількість компонентів, які відповідальні за отримання доступу для користувачів файлів, що розташовані на сервері, наприклад - HTTP-сервер. HTTP-сервер є однією з частин програмного забезпечення, що розуміє веб-адреси та

HTTP[11]. Сервери мають можливість зберігання веб-сторінки та надають їх клієнту за зверненням до нього, який оброблено через HTTP, що має головну роль по причині протоколу, що покладений для передачі інформації від сервера до клієнта, та у зворотному напрямку.

Природньо, що мережа схильна до атаки з боку хакерів, які атакують веб-сервери багатьма способами. Саме тому будь-яка вразливість в додатках, базах даних, операційних системах або в мережі обов’язково буде використана в атаці на веб-сервер.[12] Під атакою на веб-сервер мається на увазі порушення нормальної працездатності вузла мережі, видалення або модифікація його вмісту або отримання привілейованого доступу до сервера.

Беручи до уваги розмаїття технологій, що використовуються у компонентах веб-сервера, виникає необхідність в аналізі мережевих атак на веб-сервер та способів захисту від них. [13] На рисунку 4.1 наведено стек вразливості веб-сервера.

4.2 Аналіз структури системи реагування на комп'ютерні надзвичайні події

Використання вразливостей, які можуть містити веб-додатки, може привести до втрати даних, пошкодження як звичайних користувачів, так і великих компаній,тому необхідно проаналізувати вітчизняні організації, які займаються питаннямикібербезпеки, для визначення передових методів захисту веб-додатків. [14] В Україні діє команда з реагування на надзвичайні ситуації, відома як CERT-UA. CERT - UA є структурним підрозділом Державної служби спеціального зв'язкута захисту інформації України. CERT - UA займається:

– збиранням та аналізом даних про відомі кіберзагрози в державі;

– аналіз та оцінка державних інформаційних ресурсів на рівень стійкості кіберзагрозам;

– участь у реагуванні на інцидент з інформаційної безпеки;

– участь у форумах з кібербезпеки.

Варто відзначити, що, за даними веб-сайту НБУ, є інформація про можливе створення так званого "центру реагування" на загрози та інциденти у сфері

кібербезпеки в системі банків та сфері платежів України (CERT - НБУ). Також, на веб-сайті Дніпровської місцевої ради можна знайти статтю про відкриття так званого центру реагування на кібер- та онлайн-атаки в місті Дніпро, в якому працюють фахівці з IT-галузі та різних напрямків інформаційних технологій.

Варто відзначити, що у березні 2016 року Верховна рада України затвердила так звану Стратегію кібербезпеки України. Тобто, в даний момент в Україні здійснюється реагування на різного роду кібер-загрози. Планується поліпшення цією системи, за допомогою розробки нормативних документів, які пов'язані з питанням кібербезпеки та створення технологій для захисту від кібер-атак. Також був створений форум для реагування на інциденти та команди з безпеки (FIRST), метою якого був обмін інформацією про кібер-загрози. Він складається з декількох команд для реагування на різного роду надзвичайні ситуації. Було розроблено систему оцінки вразливостей системи загальної вразливості CVSS.

4.3 Опрацювання основних типів мережевих атак

4.3.1 SQL-ін’єкції

База даних це сама зручна форма для зберігання даних. Зазвичай для доступу до бази даних використовують спеціальну мову запитів – SQL, що є можливою завдяки інтерпретатора[15]. Інтерпретована мова – це мова, що включає в себе компонент часу виконання, що представляє код мови та виконує функції, які містяться в ньому[16]. Завдяки тому, як представляється мова, виникає низка вразливостей, які відомі під назвою – впровадження коду[17].

Можливість виконання впровадження, зазвичай виникає тоді, коли ненадійні дані відправляються інтерпретатору під виглядом запиту або команди. Процес при якому додаток направляє запит до сховища даних, такий самий незалежно від того ким цей запит було проведено, адміністратором програми чи привілейованим користувачем[18]. Веб-додаток працює як дискреційне управління доступом до збережених даних, створюючи запити отримання, додавання або зміни інформації у

базі даних на основі облікових засобів та типів користувачів.

Атака за допомогою ін’єкцій, буде успішною, коли вона змінює запит, а не лише дані в запиті, має можливість обходу дискреційних засобів контролю доступу до додатку та отримання несанкціонованого доступу[19]. Коли логіка програми може контролюватися результатами запиту, то зловмисник може змінити запит для зміни логіки програми.

Приклад сценарію атаки:

Первісна мета коду - створення SQL-запиту для вибору користувача з заданим ідентифікатором користувача. Користувач може ввести деякі "розумні" вхідні дані таким чином:

UserId = 123 OR 0=0

Тоді можливий SQL запит буде виглядати так:

SELECT UserId, Name, Password FROM User WHERE UserId = 123 OR 1=1;

Наведений вище SQL є дійсним і повертає всі рядки з таблиці User, оскільки OR 1 = 1 завжди TRUE. Додатки, які реалізують функцію входу в систему на основі форм у браузерах, використовують бази даних для зберігання облікових даних користувача і виконують простий запит SQL для перевірки кожної спроби входу в систему.[20]

Тобто хакер може отримати доступ до всіх імен користувачів і паролів у базі даних, просто вставивши в поле введення 123 OR 1 = 1. Для захисту від таких атак є, наприклад, перевірка вхідного SQL запита на ін’єкції, або використання ORM.

4.3.2 XSS-атака

XSS (Cross-site scripting) атака - це атака на вразливість на сервері, що вставляє шкідливу HTML розмітку або шкідливий JavaScript код в результат роботи сценарію (в дані) в тих випадках, коли сценарій не фільтрує дані, які надійшли від користувача. Через те, що фільтрація не відпрацьовує, сервер не перевіряє дану змінну на наявність в ній знаків, які не мають бути пропущені -, <, >, ', «. Код може містити шкідливу інформацію, яка може скомпрометувати комп'ютер жертви через вразливості веб-

браузера.

Також XSS може містити шкідливий JavaScript код, який надсилає свої облікові дані сеансу на інший веб-сервер[21]. Використання міжсайтових сценаріїв може бути направлено на те, щоб надати хибну інформацію жертві, наприклад, неправдиву інформацію «реального світу», таку як новини, які виглядають так, як ніби вони отримані з довіреного джерела[22]. Вміст може містити форми входу в систему, які вразі відправки відправлятимуть облікові дані для входу на веб-сервер, що належить хакеру, замість «реального» сервера. Такий тип атак виділяється тим, що шкідливий скрипт з сервера виконується на комп'ютері клієнта, причому викликає його сама жертва[23].

У цьому прикладі припускається, що кінцевою метою зловмисника є крадіжка файлів cookie жертви за допомогою вразливості XSS на веб-сайті. Додаток використовує ненадійні дані при створенні фрагмента HTML без перевірки вводу:

document.body.innerHtml = document.body.innerHtml.concat("<input name = 'cardcredit' type = 'NUMBER' value = '" +request.getParameter("ID") + "'>");

<script> document.location ='http: //www.attacker.com/cgibin/cookie.cgi?foo='+ document.cookie; </script>

Атакуючий змінює параметр «ID» в браузері.

Це приводить до надсилання даних з кукіс жертви на сайт злочинця, зокрема, ID сеансу жертви, що дає змогу захопити існуючу сесію користувача.

4.3.3 CSRF -атака

Підробка міжсайтових запитів (CSRF) - це атака, яка змушує користувача веб-ресурсу виконувати небажані дії в додатку, для якого на даний момент проводиться перевірка автентичності [24]. Такі вразливості можуть виникати, коли програмне забезпечення використовує лише cookie-файли для автеентифікації користувача, який відправив запит [25]. Оскільки файли cookie автоматично додаються до запитів у всіх браузерах незалежно від джерела запиту, зловмисник має можливість створити

шкідливий веб-ресурс, який підробляє міждоменний запит до незахищеного додатку. Зловмисник може створювати підроблені HTTP-запити і змушувати жертв відправляти свої дані через теги зображень, XSS або інші методи[26]. Зловмисник може змусити користувача виконати будь-яку операцію зі зміною стану профілю: вхід в систему, оновлення облікових даних, виконання фінансових транзакцій, тощо.

Приклад сценарію атаки кібер-злодієм.

Додаток дозволяє користувачеві відправити запит на зміну даних, який немістить ніяких секретних даних:

http://application.com/account? transfer =10000&destinationUser=5678.

Після цього зловмисник створює запит, що переводить гроші з клристувача на його власний рахунок, і додає цю атаку в запит на зображення, яке зберігається на його сайті:

<Img src = "http:// application.com/ account? transfer =10000& destinationUser =bedUser # "width =" 0 "height =" 0 "/>.

Якщо користувач відвідує сайти зловмисника після проходження аутентифікації на ресурсі example.com, підроблені запити можуть включають інформацію про сеанс, дозволяючи проведення транзакції зловмиснику.

4.3.4 Метод Broken authentication

Зазвичай, спеціалісти та інженери не звертають уваги на це надзвичайно вразливе місце або приділяють занадто мало часу, хоча саме такого типу атаки можуть привезти до того, що зловмисник легко обійде автентифікацію веб-додатка, перехопивши ID користувача або токен автентифікації. Зазвичай, в фреймворках для автентифікації та авторизації реалізоване використанням веб-додатком так званих "session cookies", мета яких є ідентифікація кожного користувача окремо [27].

Як тільки юзер вводить свій логін та пароль, веб-додаток за допомогою вище зазначених фреймворків авторизує користувача, в так званому кеші браузера створюється спеціальний ID, який надалі передається в кожному запиті на сервер веб-

додатка. Як раз, за допомогою цього ідентифікатора, сервер ідентифікує окремого юзера. Але, в ситуації, коли зловмисник перехоплює так званий "session ID" системи, всередині якої не було належної безпеки та захисту інформації, наприклад, перевірки IP-адреси у сесії або, можливо, перевірки сесії на кількість з'єднань, хакер легко може отримати доступ до облікового запису користувача. Коротше кажучи, метою зловмисників зазвичай є захоплення декількох облікових записів та отримання доступу до сервера за допомогою прав цих атакованих юзерів.

Наступні типи слабкостей можуть дозволити зловмиснику обійти методи аутентифікації:

– Вхідні дані автентифікації користувача не захищені при збереженні.

– Реєстраційні дані, які легко підібрати (наприклад, пароль «password»)

– Ідентифікатори сеансів відображаються в URL-адресі (наприклад,при копіюванні URL-адреси

– Значення сеансу не вичерпується або не стає недійсним після виходуз системи.

– Ідентифікатори сеансів не повертаються після успішного входу.

– Ідентифікатори сеансів та інші облікові дані надсилаються через незашифровані з'єднання.[28]

Сценарій реалізації атаки:

У додатку електронної комерції додається ідентифікатори сеансів до URL-адреси:

http://example.com/discount/discountitems/jsessionid=2LDM4968MS20NDW OCMJV/?item=laptop

Користувач, що вже був аутентифікованим на сайті, пересилає URL своїм друзям, щоб розповісти про знижені продажі. [29] Він надсилає електронну пошту вищезгаданим посиланням, не знаючи, що тим самим видає ідентифікатори сеансу. Коли його друзі використовують посилання, вони можуть використовують як і його сеанс ,так і його кредитну картку. [30]

4.3.5 Атака невірної конфігурації

Атака невірної конфігурації застосунку використовує недорахунки конфігурації, які були знайдені у web-додатках [31]. Саме цю вразливість завжди дуже складно віднайти. Вразливість залежить не від написанного коду програми, а віл того, яким чином обробляється налаштування параметрів у додатку [32]. Невірна конфігурація безпеки може відбудится усюди:

– платформа;
– веб-сервер;
– сервер додатків;
– база даних;
– фреймворк [33].

Саме тому неправильні конфігурації безпеки можуть різнитися як від налаштування прав доступу до облікових записів бази даних, так і до налаштування фреймворків [34].

У багатьох дадотках зараз за замовчуванням ввімкнуті деякі непотрібні та небезпечні функції. Такі як функції забезпечення якості і функції налагодження [35]. Саме таким чином хакер може отримати необхідні відомості для того щоб обійти аутентифікацію та отримати доступ до персональної і конфіденційної інформації. [36]. Окрім цього, дефолтна робота може включати в себе логіни чи імена юзерів та їх паролі, облікові записи, неправильні налаштування файлів, спеціальні механізми доступу, що доступні через веб-сервери [37]. Ризиком невірно сконфігурованної системи є те що додаток використовує базу даних, платформу, платформу додатку, мережу, містить будь-який код або ж веб-сервер. Найбільш суттєва вразливість на сьогодні є як раз невірна конфігурація. З цим погоджується вся IT-спільнота [38].

Таким чином, можна дуже часто зустріти наступні погані конфігурації, що історично прижились в центрах обробки запитів та інформації:

– конфігурації за замовчуванням, що ніхто ніколи не міняв, являють собою небезпеку через велику передбачуваність;

– неповні конфігурації, що через певний проміжок часу повинні бути змінені;

– інтуїтивно незрозумілий інтерфейс підключення до програми, що призводить до помилок через людський фактор.

При відсутності достатнього рівня розуміння, неправильне налаштування безпеки може створити багато ризиків для гетерогенних середовищ. Приклади неправильного налаштування:

– зайві порти для адміністрування, які мають відкритий стан для всіх додатків, тому додаток може наразитись на атаку віддаленого доступу;

– підключення до інтернет ресурсів, які є вихідними, що призведе до небажаної поведінки додатку;

– додатки, що застаріли, при спробі взаємодії з неіснуючими ресурсами, можлива імітація неіснуючих додатків для встановлення з'єднання.

Сценарії атак:

– можливе автоматичне встановлення консолі адміністратора, при якому акаунти за замовчуванням не змінюються, зловмисник знаходить сторінки адміністратора на сервері, та отримує доступ до паролів за замовчуванням, та може ними керувати[39];

– при конфігурації додатків, сервер дозволяє користувачам повертати інформацію про стек, які є потенційними недоліками, що в свою чергу надає додаткову інформацію про помилки;

– сервер для додатків може мати приклади додатків для використання на ньому, які не видаляються з основного сервера, ці додатки можуть мати певні недоліки, що тягнуть за собою проблеми безпеки, які можна використати для злому.

4.3.6 Sensitive Data Exposure

Існує цілий ряд вразливостей системи, які можна описати, як витік конфіденційної інформації та в цих вразливостях спільне те, що вони можуть виявляти конфіденційні дані, які зазвичай закодовані криптографічними

шифруваннями [40]. Фактично, чутлива інформація - це дані, які використовуються або змінюються з різних, зазвичай, негарних намірів.

Наприклад, це можуть бути як і ідентифікатори податків, так і дані облікового запису, або навіть дані банківського рахунку та карток. Під час введення юзером інформації в web-додатку, користувач сподівається на надійність серверу, тобто на те, що його особисті дані не отримають зловмисники або не будуть розголошені [41]. Зазвичай, в ненадійних система взагалі не контролюється шифрування або захист конфіденційної інформації. В цих системах просто зберігаються дані в базі даних, з якої запросто можна дістати інформацію різними способами, такими як SQL-ін'єкції та за допомогою інших маніпуляцій. Іноді бувають ситуації, при яких інформацію кодують за допомогою різних шифрування, проте цього може не вистачити для повноцінної надійності, так як більшість web-додатків до сих пір використовують занадто прості криптографічні алгоритми або досить таки ненадійні сховища для захисту конфіденційної інформації [42].

Взагалі, небезпека для конфіденційних даних може з'являтися, як від зовнішніх, так і від внутрішніх атак. Цікаво, що іноді найбільшої небезпеки для надійності інформації може бути від саме внутрішніх працівників компанії, адже в нього зазвичай вже є доступ до внутрішніх ресурсів компанії, тобто є можливість того, що він буду зловживати своєю посадою. В наш час дуже популярним методом надійного зберігання інформації є так звані clouds, які розташовані на різних серверах у різних постачальників хмарного зберігання, але існує вірогідність неналежної безпеки платформи, адже вона є відкритою для атак [43].

Дані вразливості є складними для використання зловмисниками, але їх вплив є значним, тому необхідно робити відповідні рішення при побудові архітектури додатку. Майже всі сучасні компанії використовують веб-програми для своєї діяльності, при цьому дані постійно перебувають як під впливом внутрішніх, так і зовнішніх загроз[44]. Даний ризик може привести до сильних витрат, що включають у себе ресурси на створення більш сильної системи безпеки, витрати на оплату регулярних штрафів. Приклад сценарію атак: Ідентифікатор можна використати для отримання всіх записів у базі даних, що не належать до них. Можна розглянути дану

адресу в незахищеному додатку: www.application.com/profile/1234.

У даній адресі наявний ідентифікатор 3032, що являє собою ідентифікатор запису у базі даних. Виходячи з того, що ідентифікатор відображається в адресі, зловмисник може змінити даний ідентифікатор, та отримати доступ до всіх профілів, що записані в базі[45].

Також одним з прикладів використання адреси для роботи з файловою системою: www.application.com/statistics?search=2021stat.doc.

В даному прикладі за допомогою параметру name вказується точне ім’я файлу для його отримання. Даний параметр можуть змінити шахраї, для доступу до файлів, до яких немає прямого доступу, але можна вказати ім’я та спокійно до них звернутись. Такі дані використовуються для виконання операції в системі: http:// application.com/PasswordChange?account=anyacc.

У даному випадку значення параметра user використовується для повідомлення додатку, та якому користувачу він повинен змінити пароль. У більшості випадків, такий крок буде частиною багатокрокової операції. Перший етап буде відповідальний за отримання запиту, про те, який пароль буде змінено, наступний крок передбачає надання нового паролю без дублювання старого[46].

Даний параметр використовується для того, щоб отримати безпосередньо пряме посилання на об’єкт користувача, для якого буде проводитися операція зі зміни паролю. Зловмисник в свою чергу може змінити ім’я користувача, що відрізняється від поточного зареєстрованого, та може отримати доступ до інших паролів користувачів[47]. Значення може використовуватись для отримання ресурсів файлової системи: http://foo.bar/showImage?img=img00011.

У таких випадках параметр file використовується для того щоб вказати додатку, до якого файлу користувач звертається та хоче його отримати. Вказуючи дані інших файлів, наприклад file = image012542.jpg, можна отримати доступ до об’єктів, що належать іншому користувачу. Такі значення використовуються для того, щоб отримувати доступ до функціональності додатку. http://foo.bar/accessPage?menuitem=12, у даному випадку значення параметра

menuitem використовується для повідомлення, до якого пункту меню користувач має намір звернутись. Але це суперечить тому, коли користувач повинен бути обмеженим тільки пунктами 1,2 та 3[48].

При зміні значення параметру menuitem є можливість отримання додаткових можливостей додатку. Можна визначити місце розташування, що відповідає за функціональність додатку, за допомогою управління пунктів меню, та виконати пункти меню, до яких спочатку він не мав доступу.

4.3.7 Вплив на безпеку системи контролю рівня доступу на рівні функцій

Вразливість можна віднести до категорії «Відсутність контролю доступу на рівні функцій» якщо недостатня або відсутня перевірка валідності в місцях, де обробляються запити. Вразливість контролю доступу на функціональному рівні може бути викликана недостатнім рівнем захисту чутливих до вразливостей місць, в яких міститься логіка обробки запитів в програмному продукті. Програма може просто закрити доступ до небезпечних дій, але не може забезпечити достатній захист, щоб запобігти певних дій або ненавмисного виконання небажаних операцій через параметр запиту керований користувачем[49].

Такі вразливості можуть бути результатом виняткових рідких випадків, помилок в базовій логіці програми і можуть бути досить складними для вирішення. Як приклад такої вразливості - неавторизований користувач, який має доступ до URL-адреси, яка може містити будь-яку конфіденційну інформацію або може надати функціональні можливості, які мають бути призначені тільки для авторизованих користувачів[50]. Іншим поширеним прикладом такої вразливості може бути просто приховування функції від небажаного користувача, але дозвіл на запит за допомогою якого користувач зможе з'ясувати, як використовувати дану функціональність.

Приклад сценарію атак:

Примусовий перегляд URL-адреси. Зловмисник змушує браузер вказувати URL-адреси. Наступні URL вимагають автентифікації.

Перейти на сайт і зверніть увагу на URL: http://abc.com/application/getappinfo

Тепер необхідно просто додати параметр, щоб побачити, чи існує сторінка. Якщо так, тепер у зловмисника є доступ з правами адміністратора до програми: http://abc.com/application/admin_getappinfo.

Права адміністратора також потрібні для доступу до сторінки адміністратора getappInfo. Якщо неавтинтифікований користувач має можливість отримати доступ до будь-якої сторінки, то це недолік веб-додатку. Якщо користувачеві без прав адміністратора дозволений доступ до сторінки admin_getappInfo, це також є недоліком і може привести зловмисника до інших незахищених сторінок адміністратора[51].

Горизонтальна атака доступу. Сторінка надає параметр для визначення функції, що викликається, і різні дії вимагають різних ролей. Якщо ці ролі не виконуються, то це недолік. Користувач переходить на сайт, реєструється для підтвердження авторизації ресурсів сайту: http://abc.com/app/userId=1 Користувач змінює UserId на інший користувач: http://abc.com/app/userId=2 Якщо належні процедури авторизації не існують, користувач тепер має можливість входити в систему як інші користувачі, просто змінюючи ID користувача[52].

4.4 Способи захисту веб серверу від атак

4.4.1 Захист від SQL-ін’єкцій

Для того, щоб запровадити захист від SQL-ін’єкцій, потрібно:

Потрібно використовувати останні доступні оновлення для систем, а також оновлювати всі компоненти, які використовуються для створення програмних продуктів, включаючи плагіни, бібліотеки, платформи на яких розробляється додаток, програмне забезпечення веб-сервера і баз даних. Також потрібно використовувати принцип надання найменшого необхідного рівня доступу для зовнішніх посиланнь та при підготовці облікових записів, які використовуються для підключення до баз даних SQL[53].

Наприклад, якщо веб-ресурсу потрібно тільки зчитати дані з бази даних за допомогою операторів SELECT, недопустимим являється надання даному обліковому записуінші привілеї на такі операції як UPDATE, INSERT або DELETE. У багатьох випадках такими привілеями можна управляти, використовуючи відповідні ролі, які існують на рівні бази даних для всіх облікових записів. Також потрібно забороняти веб-додатку підключатися до бази даних використовуючи права адміністратора (наприклад, використовувати обліковий запис «sa» на Microsoft SQL Server)[54].

Потрібно налаштувати необхідні звіти про помилки та їх обробку в коді веб-додатку і на сервері так, щоб повідомлення про помилки баз даних ніколи не були відправлені клієнту в веб-браузер. Такі дії потрібні, щоб зловмисники не мали можливості використати технічні деталі повідомлень про помилки, щоб скорегувати свої запити для успішної експлуатації ресурсу у власних цілях[55].

Також потрібно використовувати «Escape–символи» – вони дозволяють ігнорувари спеціальні символи. «Escape–символи» – це засіб вказати ядру MySQL, що одиночні лапки – це не символ, який завершує рядок, а це частина самого рядка. Дляя того, щоб MySQL знав, що даний запит безпечний, потрібно додати у відповідному місці символ зворотної косої межі.

Також можна використовувати брандмауер (WAF) для веб-додатків, які роблять запити до бази даних. Він допомагає ідентифікувати спроби SQL інжекції, а також запобігти спроб додавання інжекції SQL в додаток[56].

4.4.2 Захист від XSS-атак

Для того, щоб запровадити захист від XSS-атаки, потрібно:

Використання атрибуту HttpOnly. В сучасних веб-додатках існує такий атрибут як HttpOnly. Мета цього атрибуту в тому, що в ситуації, коли сервер налаштовує файли кукі, цей атрибут є гарантом надійності, адже можуть додаватися деякі інші атрибути, які можуть гарантувати, що файли кукі будуть надійно захищеними від різного роду шкідливих запитів, таких, наприклад, як JS:Set-Cookie: [ім'я] =

[значення]. Атрибут HttpOnly є гарантом того, що відповідь, в якій будуть зберігатися файли кукі, буде відправлена на ту ж адресу, де їх було створено.

Кодування вихідних змінних. Іноді бувають ситуації, при яких веб-додатку треба бути впевненим, що всі результуючі дані на веб-сторінці захищені, наприклад, при атаці XSS. Шифрування вихідних даних змінюється розміткою так званих "HTML альтернативних уявлень", які спеціалісти називають «entities». Під час використання «entities», веб-браузер показує результуючі об'єкти та має можливість не запускати їх відразу. Наприклад, <hello> перетворюється в & lt; hello & gt; [57].

В ситуації, коли браузер розпізнає об'єкти, вони мають змогу конвертуватись в HTML-код і можуть бути показані, проте не запущені. Наприклад, якщо хакер вставить в код <hello>alert("Hello")</hello> в HTML-код сервер повинен повернути <hello>alert("Hello ")</hello> [58].

Під час виконання браузером завантаження зашифрованого скрипта, веб-браузер конвертує цей скрипт в вище зазначений код <hello>alert("Hello ")</hello> та показує результат скрипта, як деяку частину сторінки, проте найцікавіше те, що скрипт запущеним не буде. Використання політики перетину кордонів(a crossing boundaries policy). Під час використання "crossing boundaries policy" юзери, які пройшли автентифікацію на сайті, повинні повторно ввести дані, які вони вводили при реєстрації.

Без повторного введення, користувачам не буде наданий доступ до деяких сторінок та будуть введені деякі обмеження на веб-сайті [59]. Якщо у юзера, який пройшов аутентифікацію, навіть є кукі файл, який надає змогу користувачу автоматично здійснювати вхід в систему, всеодно він буде обмежений на певних сторінках до тих пір, поки він не введе логін та пароль. Цей підхід є корисним в боротьбі з атаками XSS, адже він не дає можливості зловмиснику XSS перехопити або захопити сеанс [60].

4.4.3 Захист від CSRF –атаки

Захист від CSRF (Cross-site request forgery, міжсайтова підробка запиту) вимагає

двох речей: забезпечення того, щоб GET-запити не мали побічних ефектів, і забезпечення того, щоб не-GET-запити могли виходити тільки з клієнтського коду[61]. Щоб це забезпечити необхідно запровадити передачу стану уявлення (REST).

REST - це набір принципів проектування, які привласнюють певні типи дій (перегляд, створення, видалення, оновлення) різним HTTP-verbs. REST-фул (RESTful) дизайни будуть підтримувати код в чистому, підтримуваному стані і допоможуть масштабувати сайт. Більш того, REST наполягає на тому, щоб запити GET використовувалися тільки для перегляду ресурсів.

Відсутність побічних ефектів в GET-запитах обмежить можливу атаку, яка може бути завдана створеними URL-адресами - зловмиснику доведеться багато і старанно працювати для створення шкідливих POST-запитів[62].

Використання Anti-Forgery Tokens дають можливість визначати відправника запиту. Навіть якщо дії по редагуванню обмежені не-GET-запитами, ви не повністю захищені. POST-запити як і раніше можна відправляти на ваш сайт із сценаріїв і сторінок, розміщених на інших доменах. Для гарантії оброблення тільки дійсних HTTP-запитів, необхідно включати секретний і унікальний токен в кожну HTTP-відповідь, і щоб сервер перевіряв токен при його поверненні в повторних запитів, що використовують метод POST (або будь-який інший метод). крім GET, насправді[63].

Кожен раз, коли сервер відображає сторінку, яка виконує конфіденційні дії, він повинен записати токен захисту від підробки в приховане поле форми HTML. Цей токен потрібно включити в POST-запит на сервер (через форму або AJAX-виклики). Сервер повинен перевіряти токен, коли він повертається в повторних запитах, і відхиляти будь-які виклики з недійсним токеном[64]. Токенами проти підробки зазвичай є випадкові числа, які зберігаються в файлах cookies або на сервері. Сервер порівнює токен з вхідного запиту, зі значенням, збереженим у файлі cookie. Якщо значення ідентичні, сервер прийме HTTP-запит дійсний[65].

Перевіряти, що файли Cookies відправлені з атрибутом Cookie SameSite. Команда Google Chrome розширила заголовок Set-Cookie новим атрибутом "Same-Site", щоб запобігти CSRF. Атрибут cookie з одним сайтом дозволяє розробникам

інструктувати веб-переглядачі здійснювати контроль: чи надсилаються файли cookie разом із запитом, ініційованим доменами сторонніх виробників. Встановлення атрибута "Same-Site" у файлі cookie досить просто:

Set-Cookie: CookieName = CookieValue; SameSite = Lax; Set-Cookie:CookieName = CookieValue; SameSite = Strict;

Значення "Strict" означає, що будь-який запит, ініційований доменом третіх сторін у вашому домені, матиме cookie, видалене веб-переглядачем. Це найбезпечніше налаштування, оскільки запобігає спробам сайтів хакерів виконувати шкідливі дії з-під сеансу користувача[66]. Значення Lax дозволяє GET-запиту (і лише їм) від стороннього домену до вашого домену приєднати файли cookie. При цьому налаштуванні користувачеві не потрібно буде входити знову на ваш сайт, якщо за посиланням з іншого сайту (скажімо, результатів пошуку Google) [67].

4.4.4 Захист від Broken authentication атаки

Для того, щоб запровадити захист від Broken authentication атаки, потрібно:

Використання безпечно вбудованого менеджера сеансів на стороні сервера,який генерує новий випадковий "session ID" з високою ентропією після входу в систему. Для уникнення ненадійності та незахищеності від небезпек, ідентифікатори сеансів не повинні розташовуватись в URL-адресі. Також вони повинні ставати недійсними після виходу з системи, бездіяльності впродовж деякого часу або під час абсолютних тайм-аутів [68].

Існують спеціальні налаштування для заборони на сесійну передачу через URL, наприклад, js.insession.use_trans_sid=0; session.use_only_cookies=1; js_flag session.use_trans_sid Offjs_flag session.use_only_cookies=2.

Надійність та безпечне зберігання особистого пароля. Зазвичай, одною з найбільш складних проблем в розробці веб-додатку є надійність пароля. Правила надійних паролів робить практично неможливим вгадування пароля за допомогою різного роду засобів.

Наприклад, є політики паролів, при яких встановлюється обмеження на

мінімальну кількість символів в паролі(зазвичай, не менше ніж 10 символів), але обмеження на максимальну кількість символів не встановлюють [69]. Складність пароля. Зазвичай, механізм обробки паролів створюється таким чином, щоб він дозволяв користувачеві створювати пароль, який може складатися практично з будь-яких символів, включаючи символ пропуску.

Щоб пароль був складнішим, його треба створювати з урахуванням верхнього та нижнього регістрів. Серверний механізм зміни пароля треба створювати таким чином, щоб він був мінмального рівня складності та щоб він мав сенс, як для програми, так і для користувачів. Нижче будуть наведені приклади правил пароля:

- мінімум один рядковий символ (a-z);
- мінімум один титульний символ (A-Z);
- мінімум один спецсимвол (НЕ пунктуація);
- мінімум одна цифра (0-9);
- не більше 2 однакових символів підряд(изменено);
- пароль повинен відповідати (як мінімум) 3 з 4 правилам складності.

Заборони частовикористовуваних топологій паролів. Зазвичай, при зміні паролю на серверній частині додатку співвідносять старий та новий паролі та вимагають хоча б мінімальної зміни топології. Передача паролів тільки через TLS або інший надійний канал. Систему аутентифікації потрібно будувати так, щоб сторінка входу та, безпосередньо, аутентифікація були доступні лише через TLS. Аутентифікаційна сторінка зі входом повинна працювати через TLS-канал або, в деяких випадках, через інший надійний канал.

Потреба повторної аутентифікації для важливих функцій. Іноді виникають ризики перехоплення сеансів. Для зниження цих ризиків та ризиків CSRF, гарною практикою є вимагання від користувача повторного введення даних облікового запису перед тим, як оновлювати конфіденційні дані облікового запису, такі, як електронна пошта, телефон, пароль або якісь важливі операції [70].

4.4.5 Захист від атаки неправильної конфігурації

Для запровадження захисту від атак невірно налашотваної конфігурації, потрібно виконати наступні кроки:

– зменшуйте кількість вразливих місць за допомогою циклічних процесів;

– оновлюйте програмне забезпечення, і тримайте його в актуальному стані;

– відключайте облікові записи за заромовчанням і регулярно оновлюйте паролі;

– створюйте захищену архітектуру додатків з шифруванням даних, що мають будь-яку конфіденційну інформацію;

– переконайтеся, що параметри безпеки в бібліотеках налаштовано вірно;

– виконуйте регулярні перевірки та використовуйте інструменти для ідентифікації вразливостей в системі;

– використовуйте ту ж конфігурацію для створення, розробки і розгортання програми, оскільки відмінності це ворота для неправильних конфігурацій;

– максимально автоматизуйте систему, де це можливо, щоб уникнути помилок, що зв'язані з людським фактором;

– переконайтеся, що ваш додаток використовує останні оновлення, включаючи операційні системи, програми, системи керування базами даних та веб-сервери додатків;

– перевірте, чи немає жодних облікових записів за які можуть використовуватися за замовчуванням і чи не були змінені їхні паролі. Облікові записи за промовчанням зазвичай є джерелом проблем;

– перевірте, чи існують непотрібні або потенційно небезпечні функції. Наприклад, програма може мати функцію налагодження, яка дозволить зловмисникам обійти автентифікацію для доступу до секретної інформації;

– проводити планові перевірки безпеки додатку та сканування

вразливостей, щоб своєчасного виявлення неправильних конфігурацій[71].

4.4.6 Захист від атаки витоку критичних даних

Для того, щоб запровадити захист від атаки витоку критичних даних, потрібно:

Забезпечення шифрування даних та впровадження перевіреної техніки шифрування. Однією з найважливіших аспектів будування додатку є безпека конфіденційних даних. Саме для цього існує окрема галузь - шифрування. Проте с шифруванням треба бути обережним, обов'язково дотримуватись деяких правил.

Також потрібно шифрувати дані та визначати їх доступність, шифрувати інформацію в формі, що зберігається або в транзиті. Важливо, насамперед, зберігати конфіденційну інформацію закодованою або зашифрованою увесь час, адже такі важливі дані не повинні бути в загальному доступі і в ніякому випадку не повинні передаватися в чистому текстовому форматі. Зловмисникам дуже легко буде дістати конфіденційну інформацію, якщо вона буде у вигляді відкритого тексту [72].

Визначення даних, потребуючих додаткового захисту та обмеження доступу лише для групи підтверджених користувачів шляхом шифрування на основі ключів.

Сценарій А: Шифрування кредитної карти.

Додаток шифрує номери кредитних карт у базі даних за допомогою автоматичного шифрування баз даних. Проте, є вірогідність, що додаток автоматично буде розшифровувати інформацію після її отримання, яке може призвести до отримання номерів кредитних карток в прозорому тексті за допомогою SQL-ін'єкцій . Гарною практикою є розшифровування інформації за допомогою закритого ключа, тобто, в системі є зашифровані номери кредитних карток, шифрування яких здійснюється за допомогою відкритого ключа. Використання шлюзів безпечної аутентифікації.

Взагалі, на даний момент, використовують в додатках протокол HTTPS (SSL / TLS) - це захищений протокол, за допомогою якого конфіденційні дані, що передаються між веб-браузером та сервером залишаються приватними. За допомогою SSL можна створювати пару публічних або приватних ключів для того, щоб

передавати інформацію між браузером та сервером .

Сценарій B: SSL може не використовуватись для сторінок, які пройшли підтвердження та аутентифікацію.

Шахрай просто займається тим, що спостерігає за мережевим трафіком, як приклад у відкритій бездротовій мережі та забирає звідти cookie сессії користувача. Уже з отриманими cookie можна здійснювати злочинні дії, а саме повторити файл з даними та перехопити сеанс користувача які допомагають отримати доступ до особистих даних. Впровадження хорошого алгоритму хешування паролів. Зловмисники можуть скористатися не надійним алгоритмом для хешування паролів, щоб отримати конфіденціальну інформацію, яка збережена на сервері додатків чи веб-сервері. Для того, щоб пароль був добре захищеним є необхідність у використанні криптографічних хеш-функцій.

Сценарій C: використання несолідованих хешів для зберігання паролів у базі даних для кожного користувача.

Якщо файл пошкоджено , то зловмисник може отримати доступ до файлу паролів. Виставлення можна зробити за допомогою веселки, для хешів які не солодкі з попереднім розрахуванням. Також необхідно зазначити, що шифрування також може бути не ідеальним, використання застарілих чи слабких алгоритмів шифрування не є гарантією безпеки, а лише надає ілюзію захисту. Така ж ситуація і з простими хешами, що можна повернути. Важливо постійно оновлювати алгоритми шифрування, оскільки сучасні та сильні алгоритми, протоколи, та ключі, забезпечують високий рівень безпеки.

4.4.7 Захист від атаки відсутності контролю рівня доступу на функціональному рівні

Одна з необхідностей програм, це потреба у послідовному та простому модулі авторизації, який необхідно викликати з усіх бізнес-функціій. Як правило такий захист виконується за допомогою одного або декількох компонентів, зовнішніми до коду програми. Для безпеки рекомендовано за замовчуванням застосовувати правило

заборони, а також заборона доступу до всіх функцій програми, а вже потім надати доступ користувачам чи програмам, що його потребують. Але хоч доступ і надано, необхідно постійно його перевіряти під час звертання до ресурсу, що являє собою перевірку від дійсних користувачів, які авторизовані.

Необхідно використовувати списки контролю доступу та перевірку на наявність доступу та аутентичності. Необхідно використовувати по мінімуму привілегій та надавати доступ тільки при необхідності. Загальний доступ не повинен бути доступним, необхідно зазначити ролі для зменшення кількості прав у користувачів.

Приховувати деякі елементи через неясність не є гарним вирішення проблеми безпеки, роблячи невидимими або ж якось іншим чином, ховаючи посилання на функції інтерфейса чи кнопки, бо випадково хтось може натрапити на прихований функціонал. Окрім того, требя брати до уваги той факт, що зловмисники не будуть користуватися юзер-інтерфейсом як звичайний користувач. Скоріш за все вони будуть шукати інші шляхи відправлення запитів для того щоб надсилати їх безпосередньо, щоб отримати респонс від самої бази даних

Потрібно перевірити увесь наявний функціонал веб-додатку та URL адреси, використовуючи акаунт користувача з найнижчим рывнем доступу. Пересвідчіться що ці акаунти не можуть отримати доступ до тих частин програми, до яких вони не мають прав доступу. Зараз є певні застосунки, що надають функціонал перевірки. Вони можуть порівнювати користувачів із правами адміністратора і звичайних юзерів. Після цієї перевірки надається інформація щодо частин програми, до яких мають доступ звичайні користувачі, хоча і не повинні.

Майже всі веб-застосунки не відображають кнопки чи URL посилання для тих користувачів, які не мають прав доступу, проте варто все ж таки перевірити детальніше, бо такий метод контролю не дає стовідсоткового результату. Найкращим рішенням буде перевірка методів контроллерів і методів бізнес-логіки.

4.5 Проектування інформаційної системи

Основні можливості програми показано на діаграмі прецедентів (рисунок 4.2):

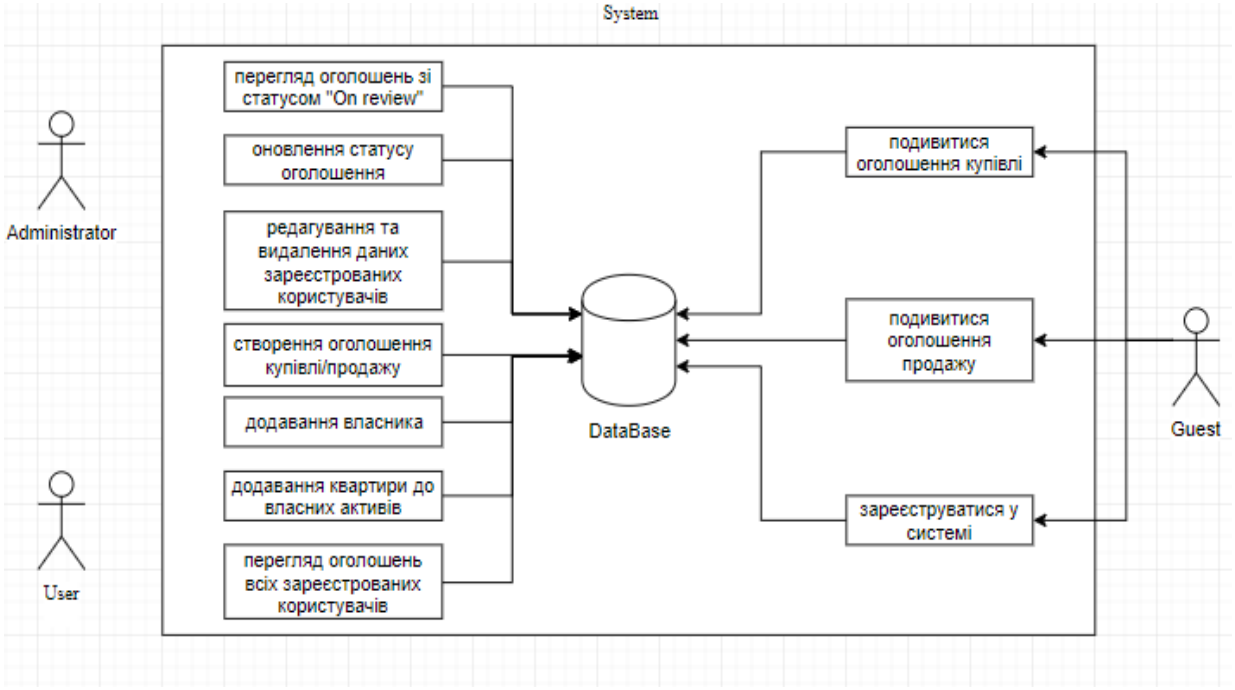

Рисунок 4.2 – Діаграма прецедентів

Під час проектування системи було віддано перевагу об'єктно-орієнтовану мову програмування C# та пов’язані технології: .NET Framework, ASP.NET Core, EF, SSMS, Swagger. .NET було обрано тому що це надзвичайно потужна платформа для розробки, яка надає можливість при необхідності легко інтегрувати інші мови, які вона підтримує, а також має багатий функціонал. Оскільки тип написаної програми WebApi, ми повинні розуміти що цей код має дуже широкий спосіб застосування, а саме: персональний комп’ютер, планшет, смартфон, сервер, веб-сайт.

Незважаючи на те що C# є C-подібною мовою програмування, тут реалізовано «збірник сміття», а саме Garbage collector. Його робота полягає в тому що він контролює роботу з пам’яттю. При необхідності він може вивільнити не задіяні ресурси для того щоб вивільняти пам’ять для наступних об’єктів. Раніше було сказано що використовувалися ці технології: .NET Framework, ASP.NET Core, EF, SSMS, Swagger.

Оскільки для роботи потрібно було розгорнути десь застосунок, було обрано

IISExpress. Так як ціллю було розробити Api потрібна була бібліотека, яка допоможе протестувати розроблений продукт. Для цього було обрано Swagger. Мова SQL використовується для побудови запитів до бази даних. Це рішення є перевірене часом і широко використовуються для розроблення схожих Api.

Для управління і доступу до баз даних використовується СУБД SSMS.

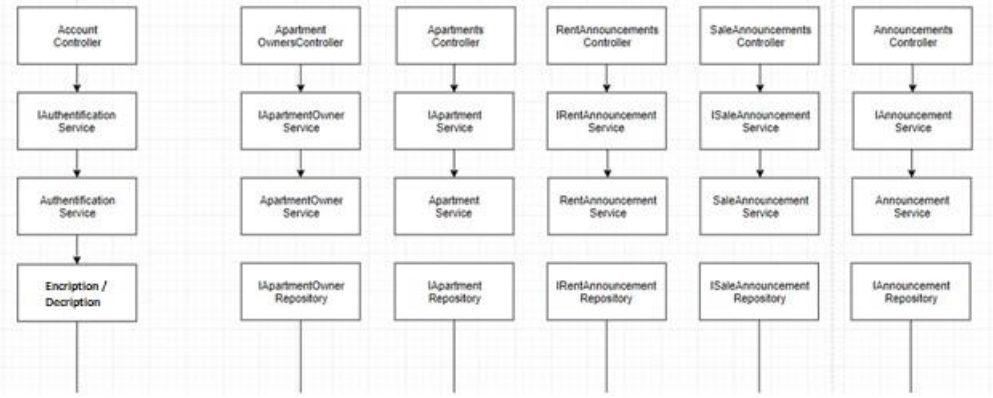

Рисунок 4.3 – Діаграма уявлення

Розглянемо діаграму уявлення (рисунки 4.3, 4.4). На рисунку зображено шість контроллерів (класи верхнього рівня): AccountController, ApartmentOwnersController, ApartmentsController, RentAnnouncementsController, SaleAnnouncementsController, AnnouncementsController.

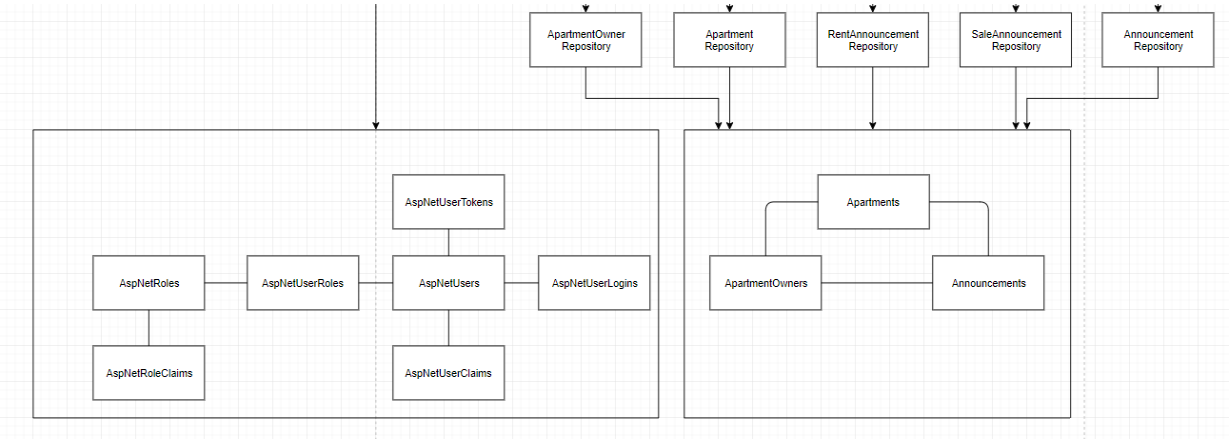

Рисунок 4.4 – Діаграма уявлення (продовження)

Дані класи є такими, що відповідають на запити користувачів, які знаходяться

на презентаційному рівні. Для того щоб обробити запит їм необхідно перейти до рівня бізнес-логіки, де зможуть звернутись до інтерфейсів, що відповідають за сервіс, що звертаються до самих сервісів. При подальшій роботі йде посилання на відповідний інтерфейс у репозиторії, що в свою чергу посилається на відповідний репозиторій. Далі репозиторії направляють запити до БД. Дана структура спроектована для забезпечення гнучкості та модульності.

4.6 Розробка бази даних

При проектуванні бази даних необхідно для початку розглянути та проаналізувати завдання. В наявності є багато інформації, яку необхідно структурувати та логічно розподілити. Для початку необхідно розуміти, що буде використано дві бази даних. Одна з баз необхідна для функціонування системи, а інша для автентифікації та авторизації. Для початку проектування бази даних необхідно вивчити предметну область задачі.

База даних необхідна для об’ємного зберігання даних про структури системи. У проекті для бази даних є окремий проект з назвою EstateEgency.DAL. Однією з вимог для даного проекту була розробка логічного та коректного найменування namespace та папок, проведено розділення проекту на папки для зручного та пришвидшеного пошуку необхідних класів. В цілому для початку було створено абстрактний клас, який буде використано для наслідування усіх сутностей. Тут знаходиться клас entity, що розташовується у папці Base, яка знаходиться у Entities. Даний клас є базовим, що описує унікальний ідентифікатор для кожної сутності. Після того як цей клас було зроблено абстрактним, реалізуються принципи SOLID, а також запобігаємо повторювання коду.

В подальшому для роботи створюються логічні сутності у папці Entities. Класом оголошення буде виступати Announcement, що зберігає в собі посилання на унікальний ідентифікатор апартаменту, власника квартири, оголошення та дату створення оголошення. Як приклад прикріплено фрагмент діаграми бази даних, на якому зображено даний клас (рисунок 4.5):

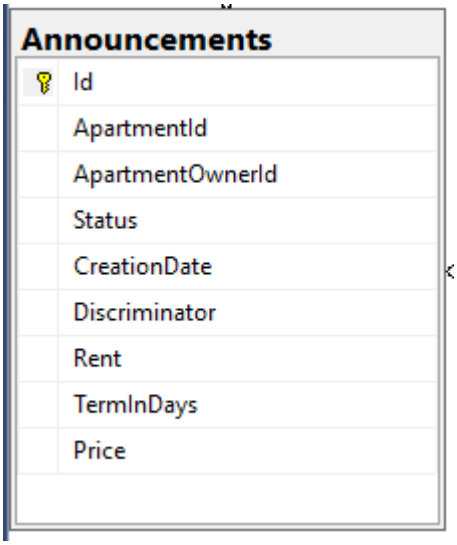

Рисунок 4.5 – Клас Announcement

Додаткова інформація про житло, в якій може бути зацікавлений користувач, зберігається у класі Apartment. Він має в собі інформацію про людину, що володіє квартирою, його унікальний номер, детальний опис квартири ті всі інші оголошення власника на цьому веб-додатку. Інформація про апартаменти може містити в собі будь-яку інформацію, починаючи від інфраструктури поруч, поверху, кількості квадратних метрів до вулиці та кількості кімнат. Діаграму баз даних наведено для прикладу, клас описано (рисунок 4.6):

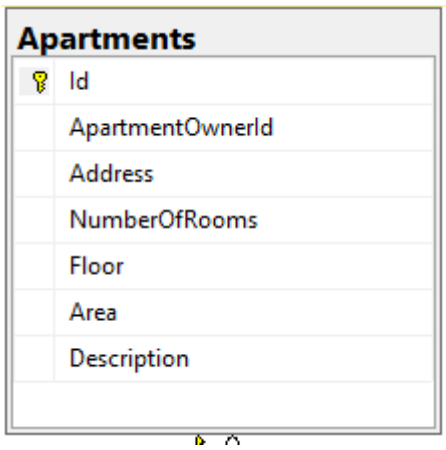

Рисунок 4.6 – Клас Apartments

Інформація про власника квартири представлена у класі ApartmentOwner. Він

утримує в собі всі об'яви цього користувача, що стосуються купівлі нерухомості або ж її продажу, а також персональну інформацію. Як правило, персональною інформацією є email користувача, його ім’я та прізвище і номер мобільного для подальшої комунікації між користувачами додатку (рисунок 4.7):

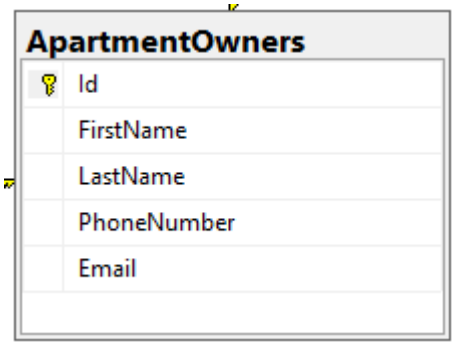

Рисунок 4.7 – Клас ApartmentOwners

Наступним кроком для того, щоб система була логічною, було додано ще два класи. Один з них - RentAnnouncement, який слугує об'єктом, котрий описує клас для того, щоб здавати в оренду нерухомість. Базовим класом, від якого наслідується RentAnnouncement, є клас Announcement, отже, всі поля та методи класу він теж наслідує. В ньому є поля, які відповідають кількості днів оренди та вартості.

Інший клас - SaleAnnouncement. Він був розроблений для ситуації, коли нерухомість не орендують, а купують. Він теж є нащадком Announcement, проте в ньому зберігається лише одне поле - ціна за купівлю. Тобто маємо таку архітектуру бази даних. Зв'язок реалізовано через відношення один до багатьох. Іншою базою даних є EstateAgencyAuthDb. В ній зберігається інформація про всіх користувачів, які були зареєстровані в системі, з різними ролями, а саме адміністратори системи та звичайні користувачі.

На жаль, в версії EF Core 3.1 не реалізоване автоматичне конструювання зв'язку багато до багатьох. Вирішенням такої проблеми є створення додаткової таблиці, так званої "розв'язуючої" таблиці, яка розділить зв'язок багато до багатьох на два зв'язки один до багатьох.

4.7 Структура проекту

Розроблюваний додаток повинен містити логіку для зберігання та доступу до даних, бізнес логіку самого додатку та логіку відображення даних. В якості архітектури для розробки такої системи було вирішено використовувати трирівневу архітектуру. Така архітектура має на меті розділити логіку додатку на три рівні - Presentation layer (рівень відображення), Business Logic layer (рівень бізнес-логіки) та Data Access layer (рівень доступу до даних).

Presentation layer – це частина системи, яка відповідає за відображення даних для користувача та обробку дій користувача. На цьому рівні розміщено інтерфейс користувача разом з розробленою системою для отримання даних від нього. Так, наприклад, технологія ASP.NET MVC реалізує на цьому рівні розміщення візуальних компонент які являють собою інтерфейс користувача (стилі, статичні сторінки HTML, JavaScript), так само і уявлення, контролери, об’єкти контексту запиту. Рівень відображення реалізовано у проекті EstateEgancy.API.

Business Logic layer – тримає в собі набір компонент, які відповідають за валідацію отриманих від рівня презентації даних, реалізує всю необхідну логіку додатку, усі розрахунки, взаємодіє з рівнем доступу до даних та передає на рівень презентації результати обробки. Ріень бізнес-логіки реалізовано у проектіEstateEgancy.BLL.

Data Access layer (рівень доступу до даних) – відповідає за доступ до даних, інкапсулює усю логіку взаємодії з даними, надаючи інтуїтивний інтерфейс до колекцій даних. Зберігає моделі, що описують сутності. Також тут розміщуються специфічні класи для роботи з використовуваними технологіями доступу до даних, наприклад, клас контексту даних Entity Framework. Тут також зберігаються репозиторії (Repositories), до яких доступається бізнес-логіка. Рівень доступу до даних реалізовано у проекті EstateEgancy.DAL.

При цьому слід зауважити, що, відповідно до трирівневої архітектури, крайні рівні не можуть взаємодіяти між собою, тобто Presentation layer не може напряму звернутися до Data Access layer. Presentation layer має взаємодіяти тільки з Business

Logic layer. Під час написання Data Access layer було також реалізовано шаблони проектування Repository (сховище) та Unit of Work (одиниця роботи).

При проектуванні системи було використано патерн репозиторій, перекладається як сховище. Це певна прокладка, яка дозволяє взаємодіяти з данними, що зберігаються у базі даних. Контракт репозиторію дуже схожий на колекцію. Це проявляється у тому, що з репозиторієм можно працювати за дуже схожими принципами додавання та видалення об’єктів як із звичайною колекцією. Загалом репозиторій варто розуміти як єдину точку доступу до баз даних, що може змінюватися в залежності від необхідних до повернення даних.

```
public class ApartmentDto
{
    0 references
    public int Id { get; set; }

    3 references
    public string OwnerFirstName { get; set; }
    3 references
    public string OwnerLastName { get; set; }
    3 references
    public string OwnerPhoneNumber { get; set; }
    3 references
    public string OwnerEmail { get; set; }

    0 references
    public string Address { get; set; }
    0 references
    public int NumberOfRooms { get; set; }
    0 references
    public int Floor { get; set; }
    0 references
    public double Area { get; set; }
    0 references
    public string Description { get; set; }
}
```

Рисунок 4.8 – Базова ДТО

Також було використано патерн одиниці роботи, що в перекладі unit of work. Принцип цього патерну полягає в тому що вирішуються деякі проблеми паралельного доступу до даних. Оскільки паралельний доступ може суттєво вплинути на програму в негативному плані, варто контролювати запити. Саме цим і займається цей патерн. Фактично він реалізує механізм транзакцій, тобто забезпечує доступ до бази даних з урахування проблем ACID, які могли б виникнути без нього. Зазвичай коли закінчується використання unit of work, ми зберігаємо статус роботи.

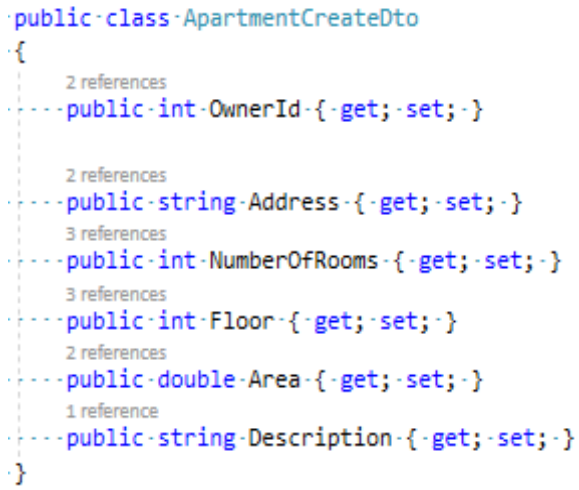

Рисунок 4.9 – ДТО для створення об’єкту

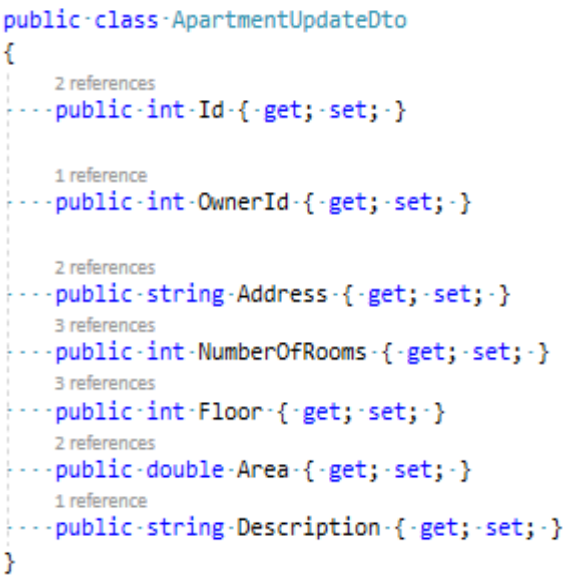

Рисунок 4.10 – ДТО для змінення об’єкту

Ключову роль займає DTOs (Data Transfer Objects), об’єкти за допомогою яких дані транспортуються на різні рівні. Для прикладу можна розглянути, що до клієнта надходять дані, які він отримує прямо з бази даних. Але як правило це не є гарною практикою. Іноді необхідно змінити форму представлення даних, які направляються до клієнта. Як приклад:

– видалення циклічних посилань;
– приховування властивостей, які повинні бути недоступні для клієнта;

– видалення певних властивостей, для зменшення корисного навантаження.

Для таких дії необхідно визначити DTO – об’єкт передачі даних. DTO вирішує які дані будуть відправлені у мережу, для прикладу наведено дані з програми (рисунки 4.8, 4.9, 4.10).

Як бачимо, нам не потрібні усі дані моделі для конкретних методів, бо вони мають різний функціонал.

Як вже зазначалося раніше, кожен користувач має власний профайл, що містить інформацію о зареєстрованому користувачеві: прізвище та ім’я, номер телефону, адреса почтової скриньки та сума на балансі. Ці дані є приватною інформацію користувача і потребують захисту, так як при потраплянні в довільне користування 3-х осіб може бути завдано шкоди, втручання в приватне життя а також стає можливою маніпуляція із грошовим балансом користувача.

За допомогою симетричного алгоритму блочного шифрування Rijndael (Advanced Encryption Standard), розроблено метод доступу до приватної інформації користувача.

RijndaelManaged cryptography class має можливість кастомної настройки типу шифрування:

```
using (var symmetricKey = new RijndaelManaged())
   symmetricKey.BlockSize = 256;
   symmetricKey.Mode = CipherMode.CBC;
   symmetricKey.Padding = PaddingMode.PKCS7;
```

При реєстрації користувачеві видається ключ для зчитування текстового представлення (документа) з його приватними даними та рахунком на балансі. Для того, щоб отримати не спотворене представлення даних, користувач повинен ввести свій ключ. Таким чином, тільки користувач з одним правильним ключем може отримати доступ до цих даних. При сценарії втрати ключа, користувач повинен звернутися до адміністратора, для того щоб згенерувати та отримати новий ключ.

4.8 Тестовий приклад роботи програми

Перед початком тестування необхідно виділити те, що система має три ролі: гість, користувач та адміністратор. Кожен з яких має свій особистий рівень доступу, найбільше повноважень має роль адміністратора, а саме відсутність обмежень. А як для порівняння, у гостя правил майже немає, він може лише проглянути оголошення про оренду чи продаж.

Користувачі ресурсу мають доступ до розміщення своїх оголошень, але самі оголошення стануть доступними для перегляду тільки після того, як адміністратор розгляне заявку. Користувачі не можуть редагувати та видаляти свої аккаунти, такі дії доступні лише адміністратору.

На рисунку 4.11 зображено вигляд вікна сайту для користувача. Для входу в систему чи реєстрацію в ній, необхідно перейти на вкладку Account. Поля для вводу даних мають валідацію, тому поки дані не будуть відповідати поточним вимогам, користувач не зможе перейти до наступного кроку.

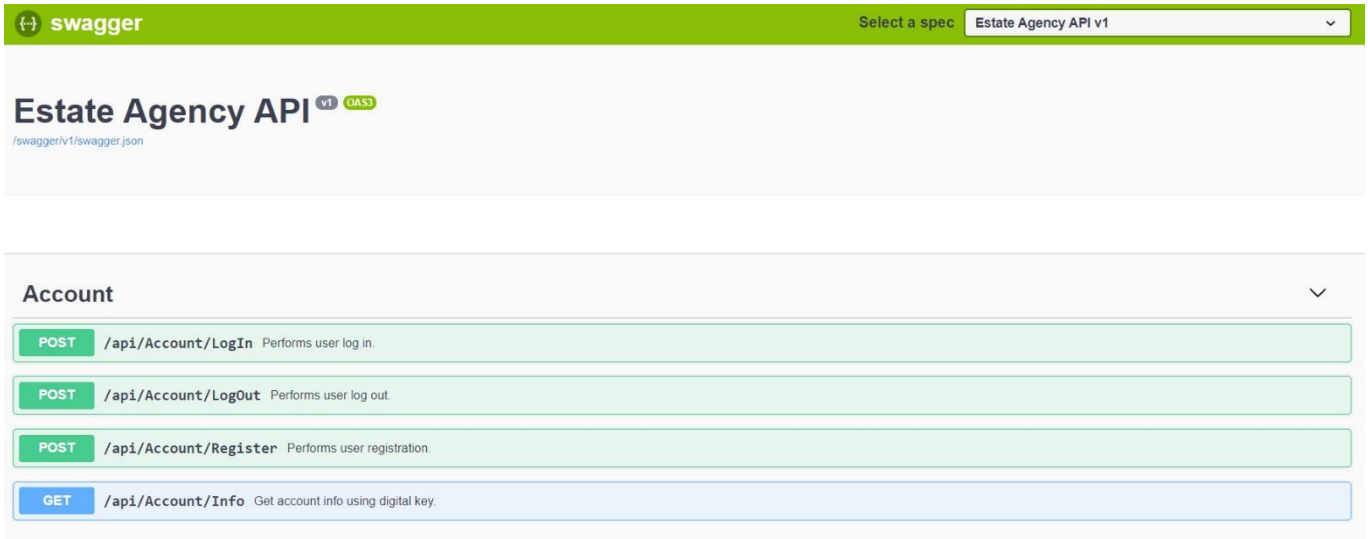

Рисунок 4.11 – Початковий вигляд сайту

Щоб авторизуватися у системі, необхідно ввести пошту та пароль, які були введені при реєстрації. Система створена для комфортного користування. Якщо

користувач ввів все правильно, без помилок, то сервер поверне код 200, тобто запит пройшов успішно. Якщо сервер повернув код 400, то щось було неправильно введено. При відповіді з кодом 500, буде означати, що щось не так із сервером. Також при реєстрації передбачений контроль за вводом користувача у форму для реєстрації.

Також необхідний є особистий кабінет користувача. В ньому користувач має доступ до свого особистого профілю, там він може редагувати свої дані. Це стандартна та загалом прийнята практика. В такому кабінеті користувач може міняти особисті дані, відстежувати свою статистику, змінювати зображення профілю, задавати параметри відображення тощо. В даній реалізації відсутній особистій кабінет користувача. Проте, завдяки дотриманню принципів SOLID та багатошаровій архітектурі програми додатку, можна буде з мінімальними зусиллями розробити та додати до проекту кабінет користувача.

Необхідно окремо відмітити, що існує ряд обмежень для адреси електронної пошти. Пошта повинна складатись з:

— літери англійського алфавіту великого регістру (A-Z);
— літери англійського алфавіту маленького регістру (a-z);
— цифри від 0 до 9;
— спеціальні символи такі як ~!@#$%^&*_-+=`|\(){}[]:;'<>,.?/.

Електронна пошта також повинна закінчуватись на, наприклад, @gmail.com.

Схожу ситуацію можна спостерігати і з паролем. Він повинен мати:

— літери англійського алфавіту великого регістру (A-Z);
— літери англійського алфавіту маленького регістру (a-z);
— цифри від 0 до 9;
— спеціальні символи такі як ~!@#$%^&*_-+=`|\(){}[]:;'<>,.?/.

Для прикладу дивіться рисунок 4.12.

Далі наведено форму реєстрації. Для цього необхідно перейти на вкладку Account. Там буде Post метод LogIn. Натиснути кнопку TryItOut. Після цього система запропонує ввести дані у дві існуючі форми: email та password.

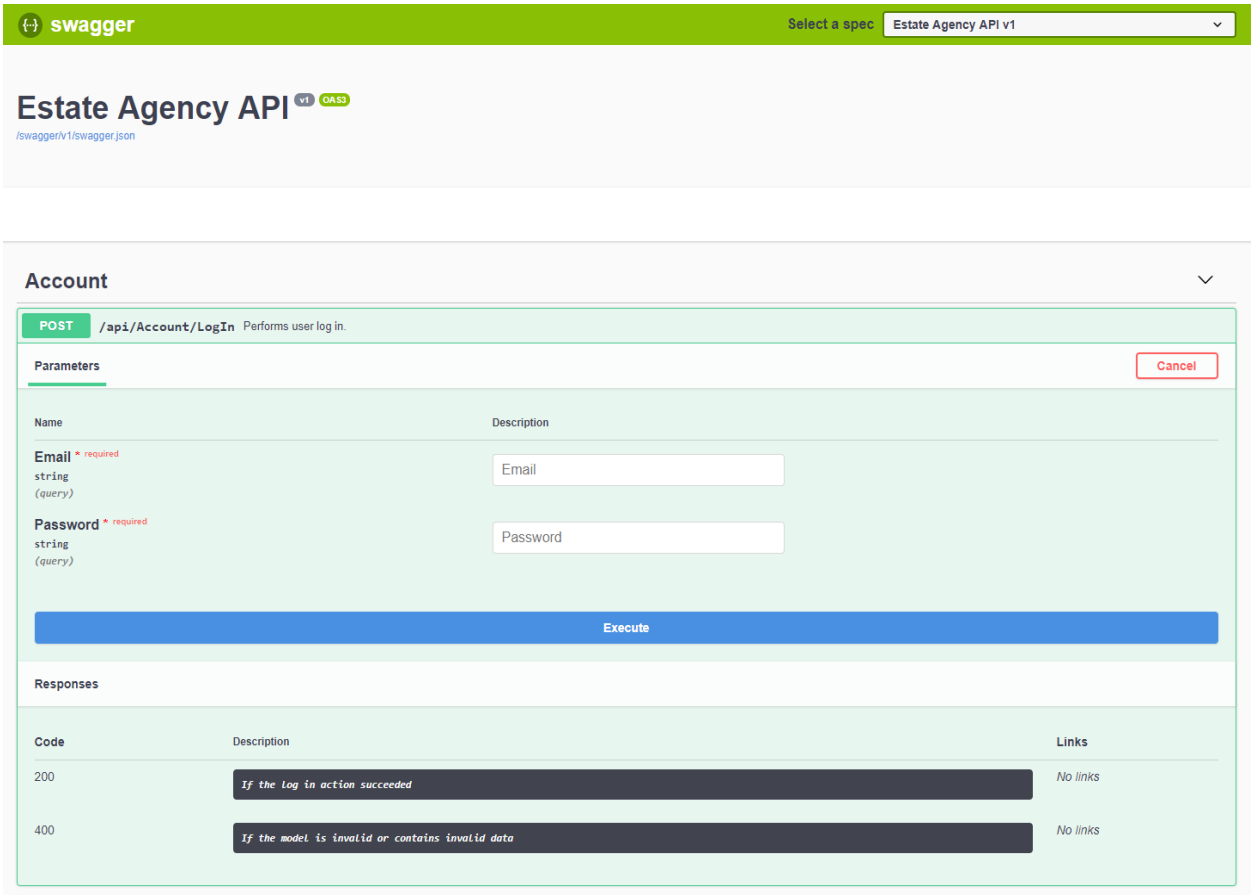

Рисунок 4.12 – LogIn

У випадку якщо вхід у систему пройшов успішно, повернеться код 200.

У випадку, коли модель невірна чи коли приходять невірні дані, система поверне 400 код.

Окрім того, на даній вкладці доступна функція реєстрації. Загалом, вкладка реалізує стандартні функції автентифікації та авторизації.

Також на цій вкладці є можливість виходу з системи. Цю функцію забезпечує метод LogOut. Потрібно лише натиснути кнопку TryItOut. Далі слід натиснути Execute. Після завершення обробки запиту натискання на останню кнопку, користувач вийде з системи, як представлено на рисунку 4.13:

Також система надає інтерфейс для реєстрації користувача. Метод називається Register. Метод реєстрації користувача характеризується наступними інваріантами:

— email – це поштова скринька, яка в подальшому використовується для взаємодії з користувачем системи. Цей параметр має бути унікальним в системі (неможливо створити два різних користувача на одну поштову скриньку);

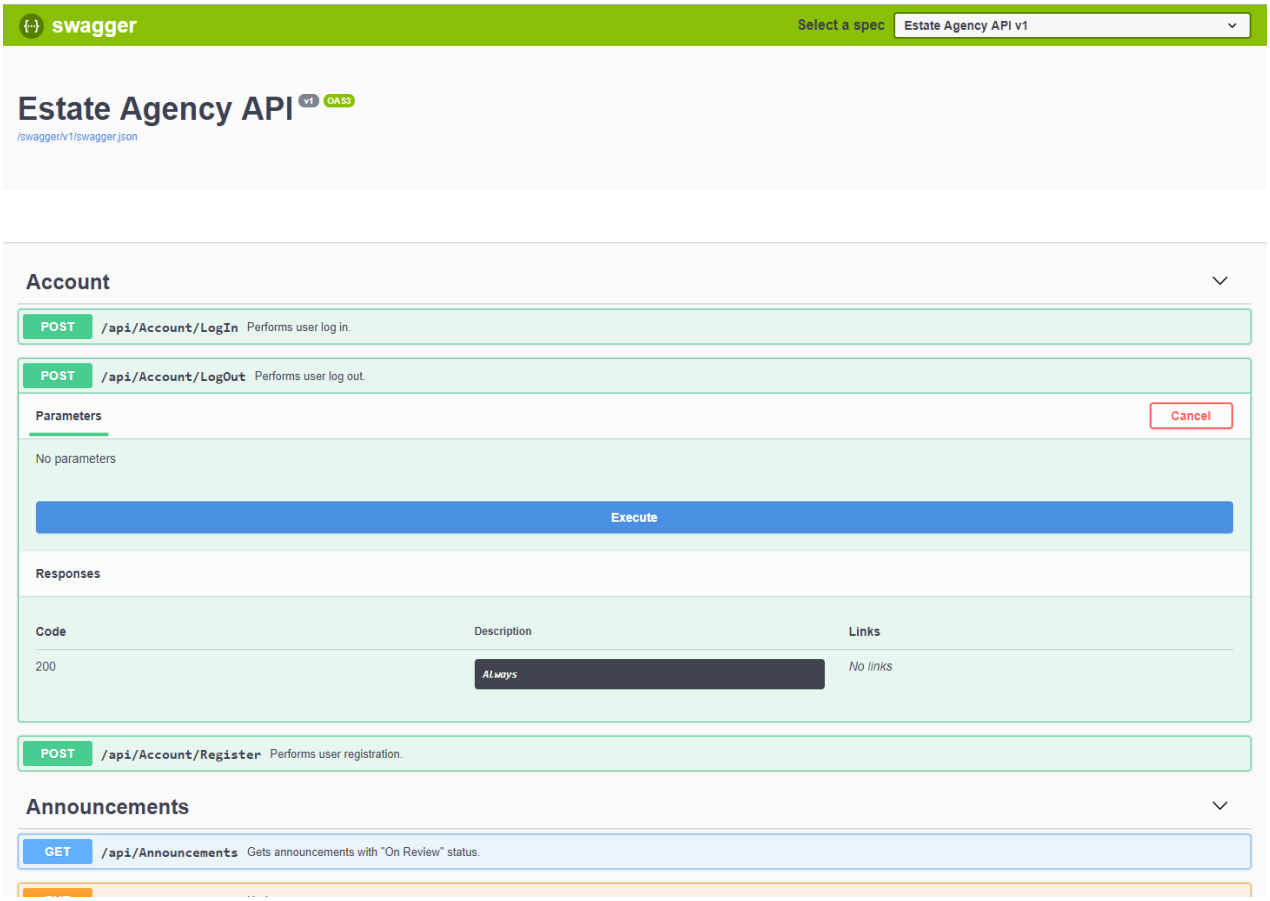

Рисунок 4.13 – LogOut

— password – це персональний пароль користувача, за допомогою якого він зможе автентифікуватися. Пароль має бути не пустим;

— passwordConfirm – поле для підтвердження паролю. Поле вводить користувач для реєстрації на одному рівні з паролем, на випадок помилки вводу;

— roles – поле, яке зберігає в собі значення ролей, які можуть бути призначені користувачеві. Як відомо, у системі їх три. Проте тут ми зможемо визначити два, бо у третьому немає потреби. Тому є сенс позначити “admin” та “user”. Приклад інтерфейсу для реєстрації нових користувачів представлено на рисунку 4.14:

На рисунку 4.15 зображено приклад отримання персональних даних користувача по роуту Get api/Account/AccountInfo. Валідні дані буде повернуто лише при передачі правильного ключа. В противному випадку, дані будуть невірно дешифруватися и повернуться в зпотвореному вигляді.

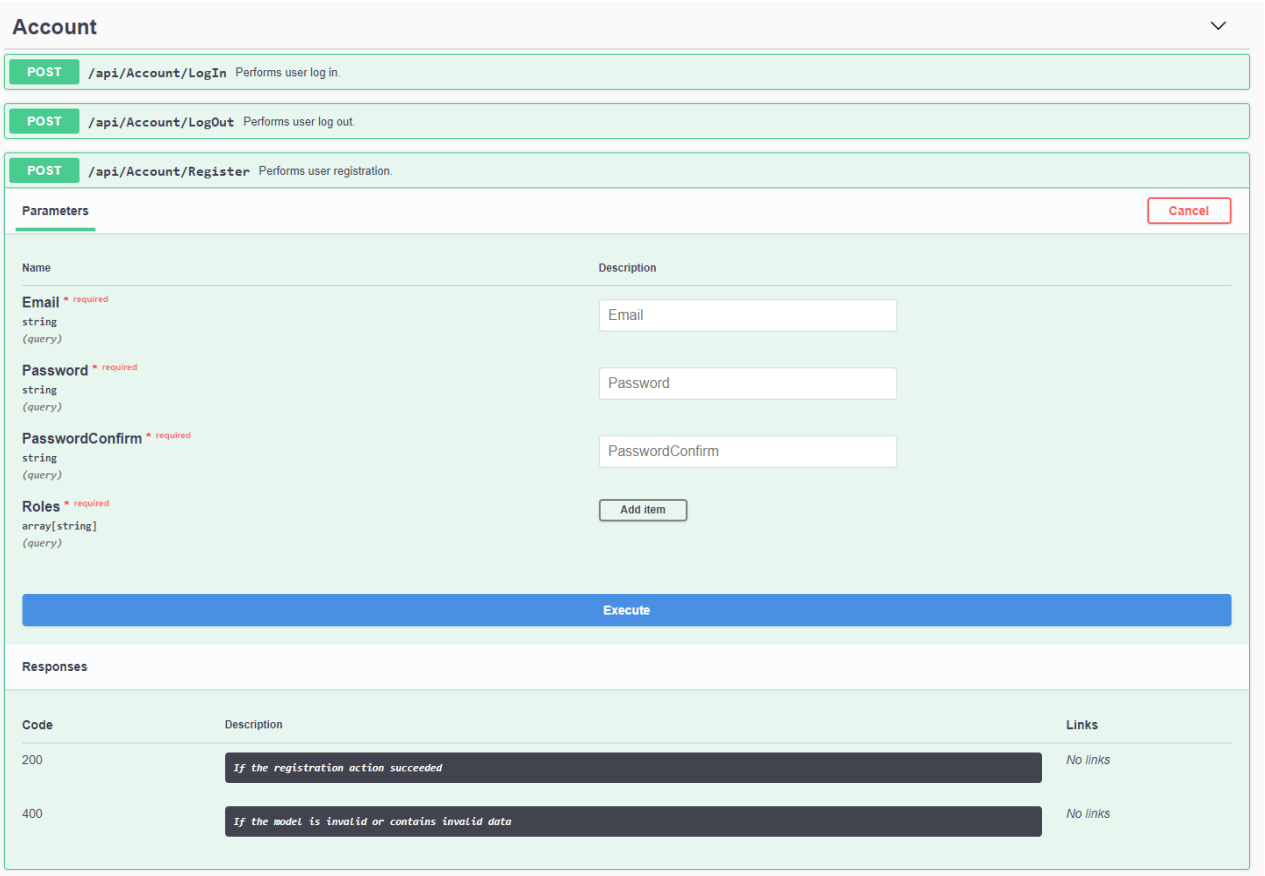

Рисунок 4.14 – Форма реєстрації

Гість може подивитися, які є доступні оголошення на оренду квартири. Для пошуку реалізовано декілька форм, які мають основні критерії. У випадку, коли їх полишити пустими, будуть показані всі можливі варіанти (рисунки 4.16 і 4.17).

Рисунок 4.15 – Отримання персональних даних користувача

Тобто, залишений без критеріїв пошук відповідає функції GetAll, що повертає всі сутності відповідного типу з бази даних. Це відбувається через реалізацію сервіса ExpressionComposer. Він об’єднує в собі всі параметри пошуку в єдиний вираз пошуку.

В разі, коли всі параметри пошуку пусті або відсутні, ExpressionComposer ініціалізується виразом «повернути все» (тобто вже на нього накладаються інші обмеження, при наявності параметрів пошуку). Цей механізм реалізовано за допомогою дерева виразів. Щоб створити дерево виразів за допомогою API, використовується клас Expression. Цей клас містить статичні заводські методи, які створюють вузли дерева виразів певних типів, наприклад, ParameterExpression, який представляє змінну або параметр, або MethodCallExpression, який представляє виклик методу. ParameterExpression, MethodCallExpression та інші типи виразів, також визначені у просторі імен System.Linq.Expressions. Ці типи походять від абстрактного типу Вираз.

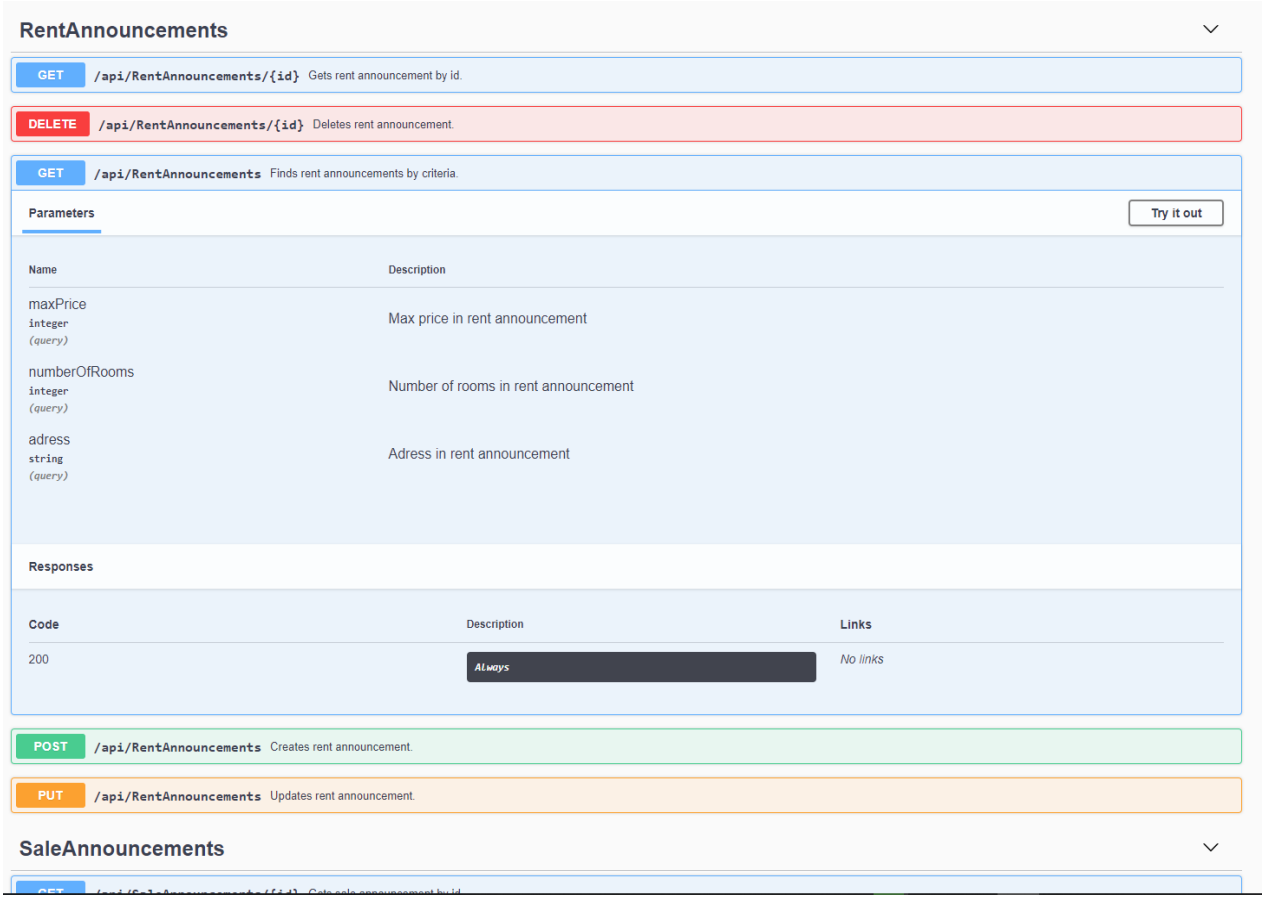

Рисунок 4.16 – Форма пошуку оренди

Як можна побачити ми маємо три параметри: адреса, кількість кімнат та максимальну ціну. Дані параметри можуть налаштовуватися для кожного користувача окремо, в залежності від персонального побажання.

За допомогою сервісів типу ExpressionComposer реалізований спосіб додавання будь-якої кількості параметрів до виразу пошуку. При розширенні функціоналу і, відповідно, параметрів пошуку, їх можна легко додати до спільного виразу пошуку, адже реалізація ExpressionComposer пробігає по списку виразів для пошуку та робить з них один. Для додання нового параметру достатньо просто занести його до списку виразів пошуку.

Якщо залишити поля фільтра пустими, то за запитом буде отримано усі можливі варіанти, які відповідають пустим фільтрам.

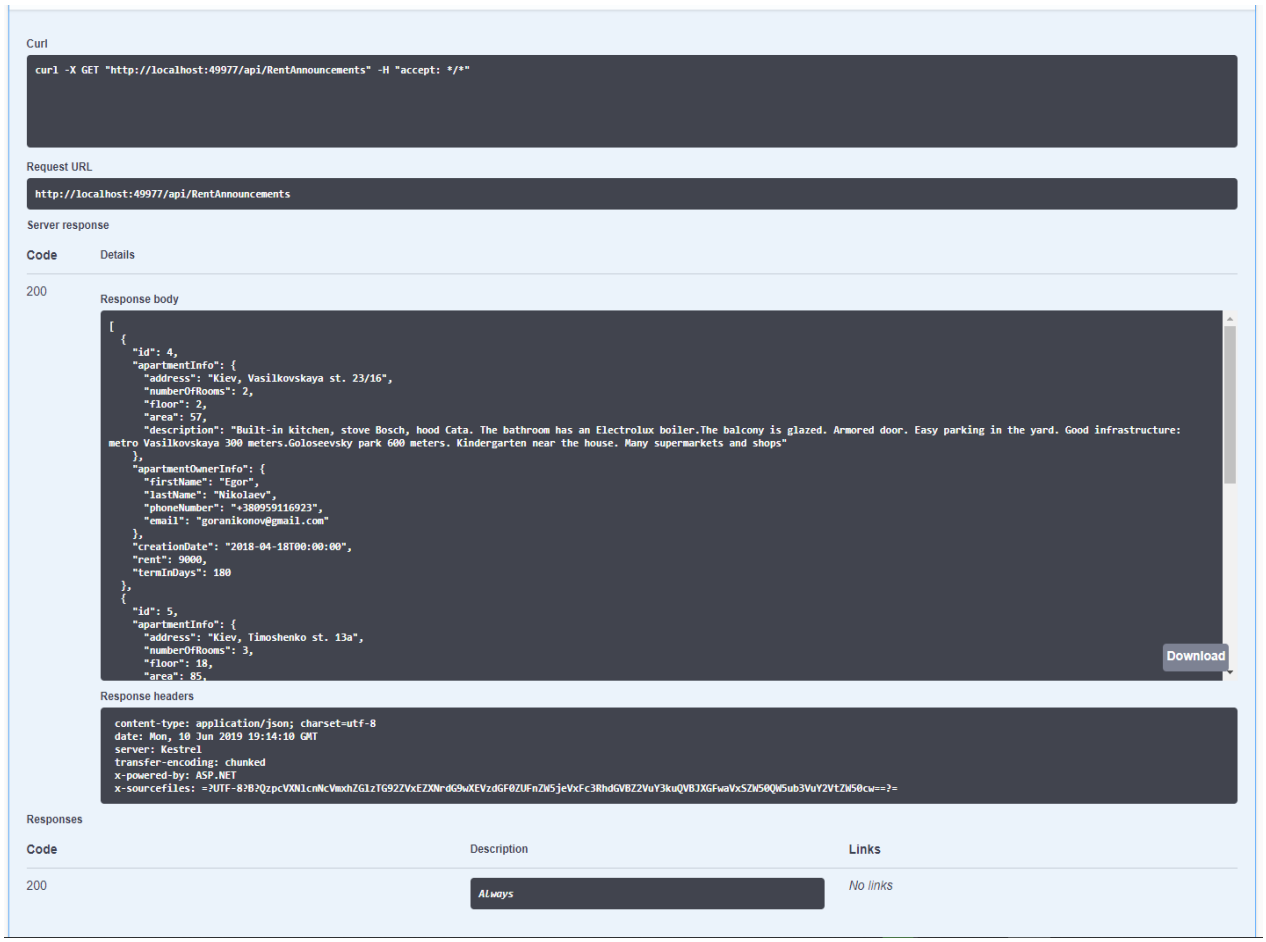

Рисунок 4.17 – Результат роботи пошуку оренди

По такому ж принципу працює пошук квартир на продаж, де гість також може за певними критеріями переглянути варіанти які його цікавлять.

Для того щоб розглянути які з оголошень знаходяться у стані перевірки, необхідно зайти під профілем адміністратора та перейти за посиланням announcements -> Get, як зображено на рисунку 4.17.

Контроллер та сервіс типу Announcements створені спеціально для доступу до функціоналу, що відповідає за зміну або перегляд статусів оголошень. Це було можливо реалізувати через те, що в базі даних оголошення, як на продаж так і на аренду, знаходяться в одній таблиці, тобто доступ до даних по всім оголошенням відбувається через спільну точку доступу (репозиторій Announcement).

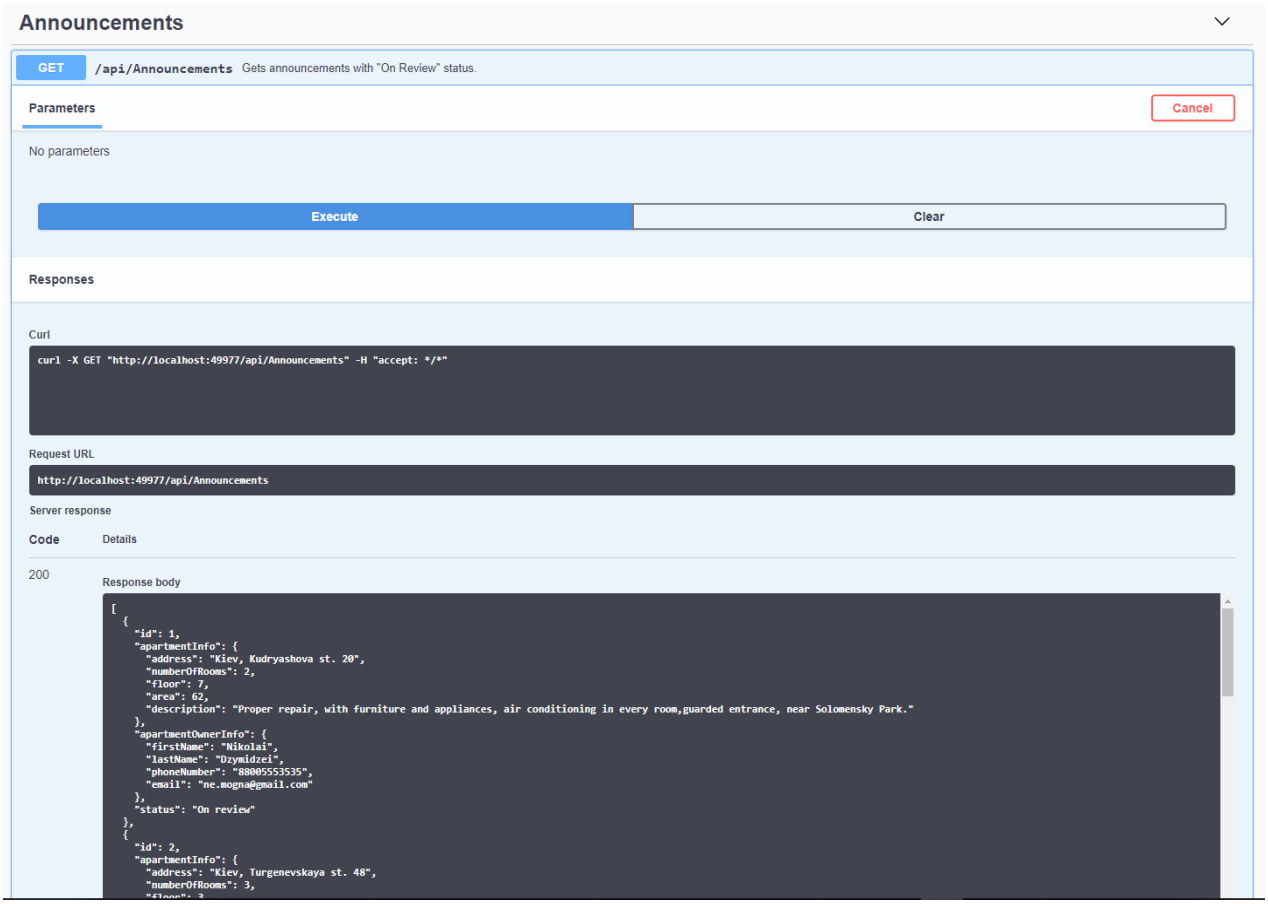

Рисунок 4.18 – Непідтверджені оголошення

Перш ніж об’яву буде додано до загального списку, який можуть переглядати потенційні покупці, вона надходить на перегляд адміністраторам (рисунок 4.18). Вкладка Announcements містить метод, який повертає нам усі оголошення, що помічені як “On review”.

Саме це і є не переглянуті об’яви. Це контроль якості створюваних об’яв. Тобто, у випадку коли адміністратор не має зауважень до поданого оголошення, він змінює його статус. У випадку, коли адміністратор не переглядає нові подані оголошення, можливе потрапляння неочікуваних системою та нормами описів на сайті через нове оголошення.

Якщо оголошення відповідає нормам сайту та вимогам адміністратора, останній змінює статус об'яви (рисунок 4.19):

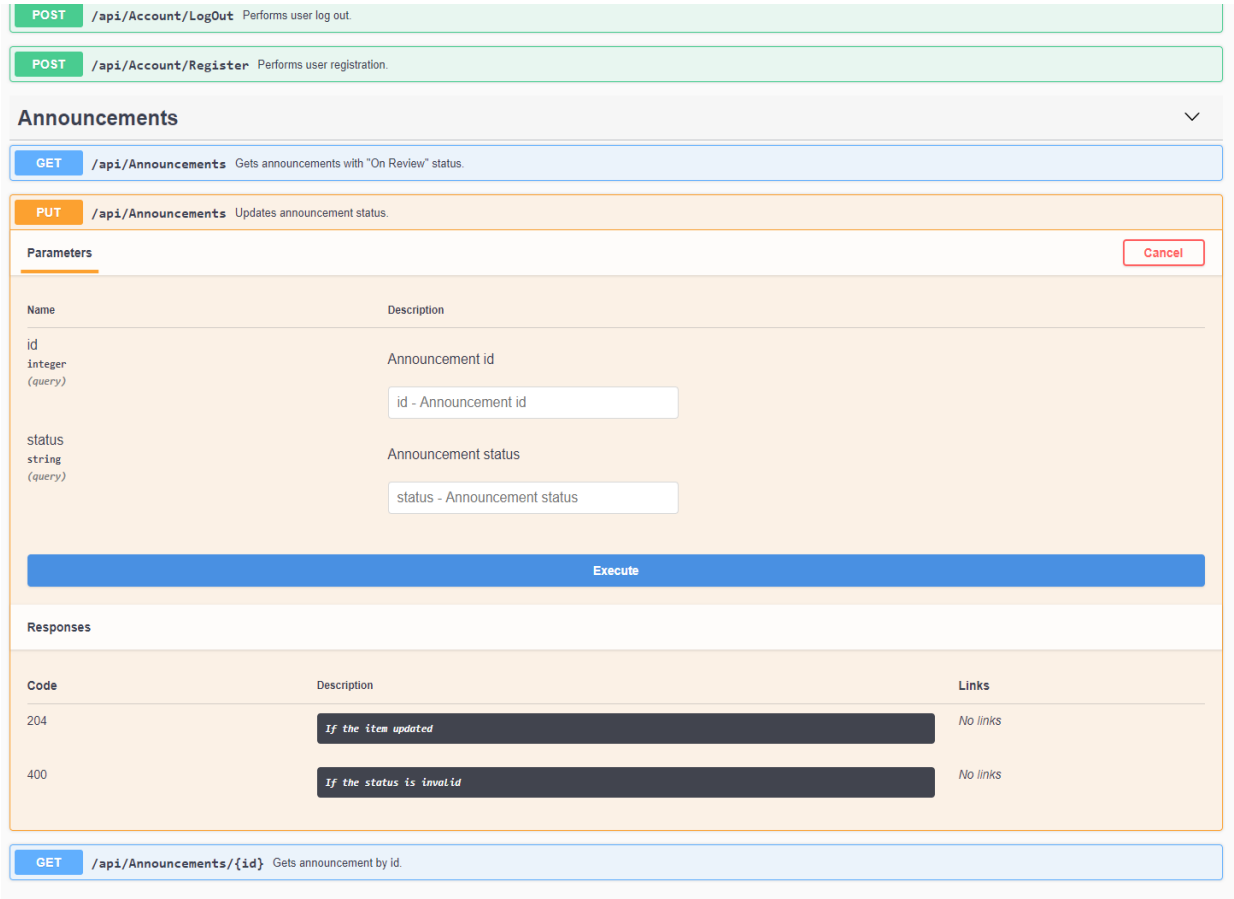

Рисунок 4.19 – Змінення статусу об'яви

Тобто, адміністрація сайту самостійно (через певні проміжки часу) повинна продимлюватися всі доступні оголошення зі статусом «On review» та вирішувати, чи дозволяти потрапляння оголошення на сайт (approve), чи відмовляти власнику оголошення (reject) та пояснити, додавши в опис, через які саме причини оголошення не пройшло валідацію. Якщо цього не робити, то нові оголошення будуть створюватися автоматично зі статусом “On review” та ніколи не попадуть на головну

сторінку сайту, адже саме для цього реалізовано механізм зміни статусу. За замовчуванням, статус «On review» не відображається в списку доступних оголошень.

Інколи виникають ситуації, в яких необхідно подивитися не всі оголошення, статус яких “On review”, а лише одне. Для цього реалізовано окремий метод Get (рисунок 4.20):

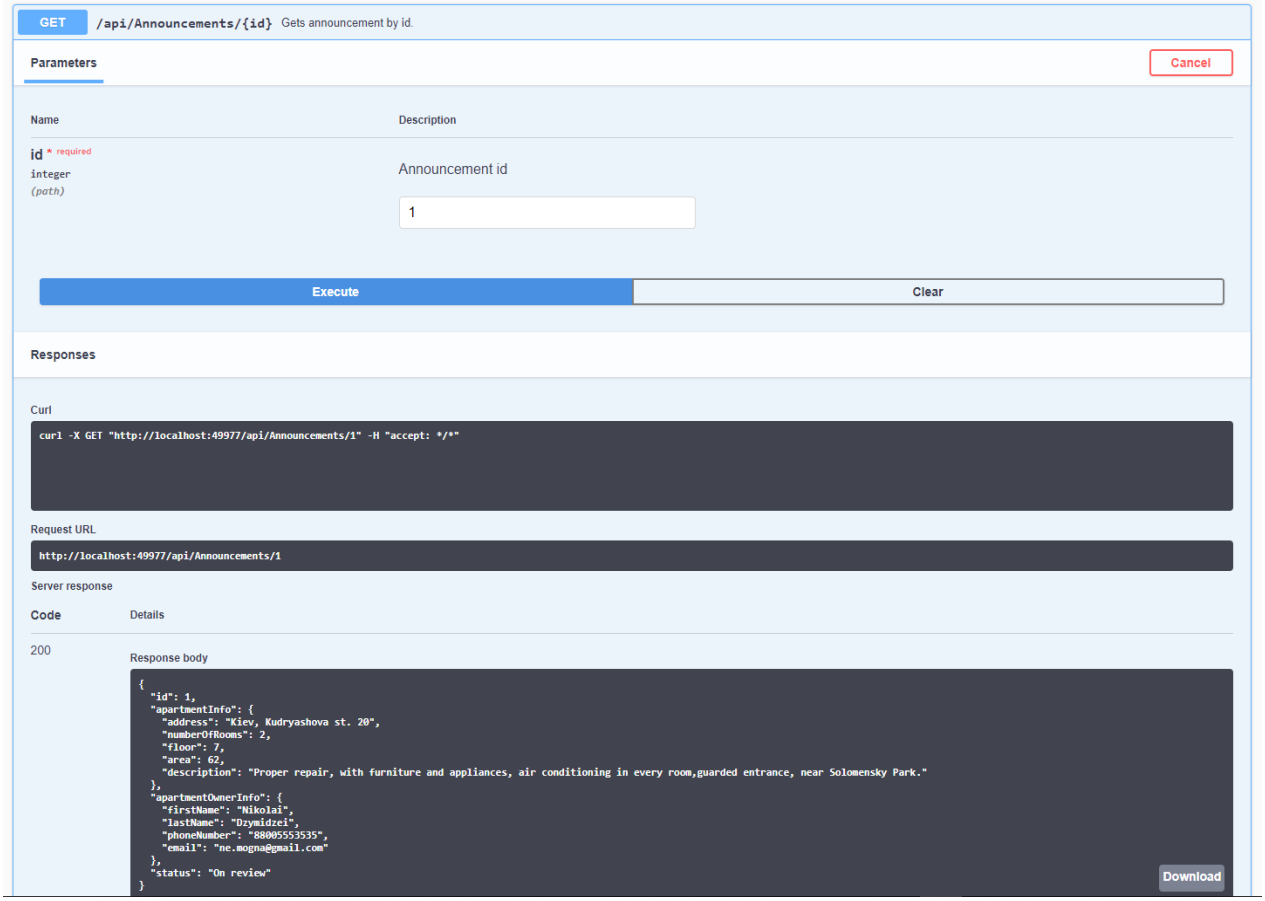

Рисунок 4.20 – Інформація по номеру оголошення

Він повертає модель лише в одному екземплярі, яку наповнює інформацією про об'єкт з визначеним ідентифікатором, який являється унікальним. Модель містить в собі інформацію про власника і його апартаменти. Стосовно апартаментів існують наступні дані:

— address – адреса квартири;

— numberOfRooms – кількість кімнат;

— floor – поверх;

— area – площа апартаментів;

— description – додатковий опис квартири.

В останньому пункті може бути вказано наприклад технічне оснащення квартири, переваги її розташування, інфраструктура, наявність поруч магазинів, аптек, шкіл, метро, дитячих садків тощо.

Екземпляр власника містить в собі наступні дані:

— firstName – ім’я;

— lastName – прізвище;

— phoneNumber – номер телефону;

— email – електронна адреса власника.

Номер телефону та адреса електронної скриньки необхідні для встановлення контактів з власником даної квартири та обговорення в телефонному режимі додаткових питань на рахунок купівлі/продажу квартири.

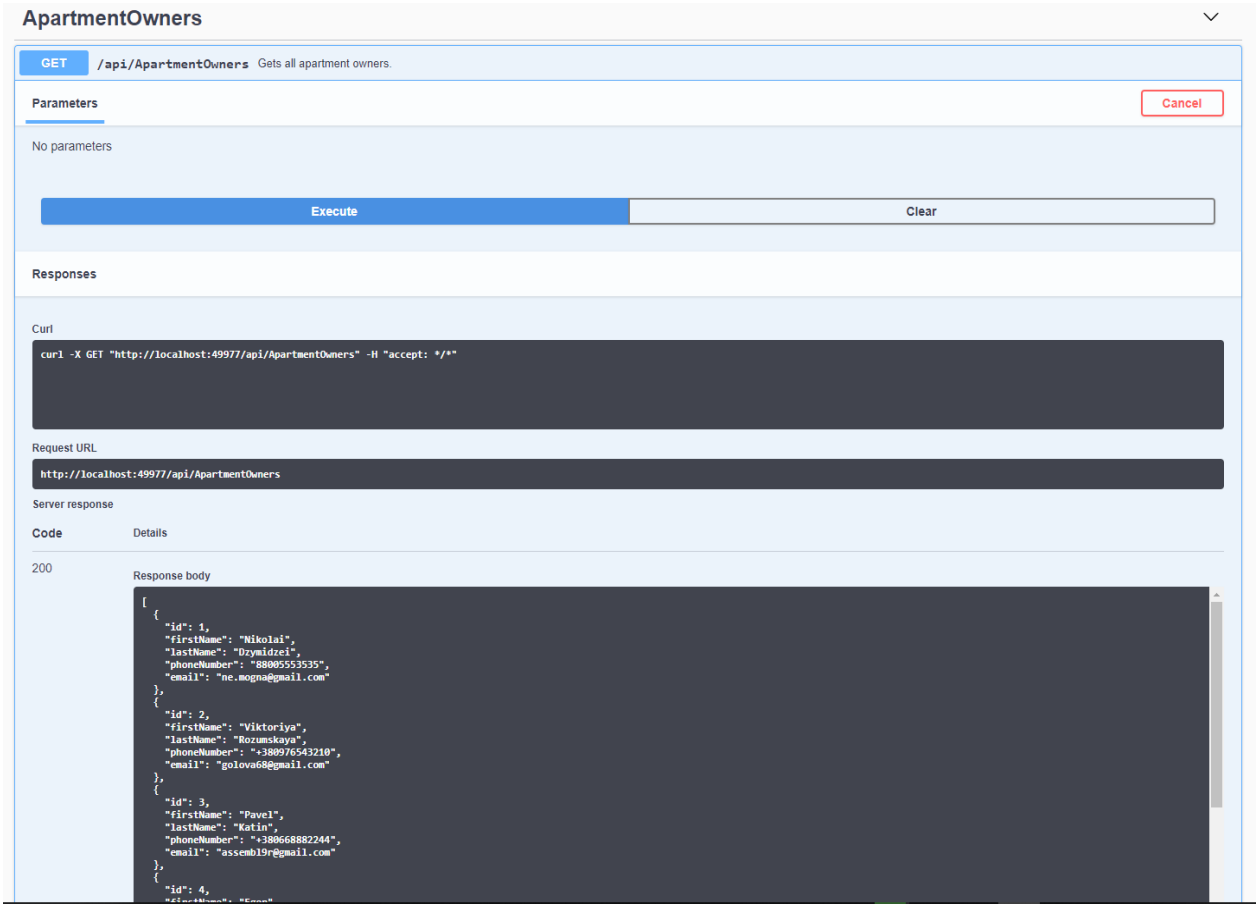

Рисунок 4.21 – Власники квартир

Для прикладу можна розглянути власників квартир. Даний запит повертає усіх

власників, які зареєстровані в системі. Серед даних що повернулися є інформація щодо їх унікального ідентифікатору, прізвища, імені, номеру телефону та поштової скриньки кожного для подальшого контакту з ними (рисунок 4.21):

Тут можна розглянути яким чином відбувається робота з додавання користувача, як власника квартири. Для цієї реалізації необхідно заповнити форму, що містить інформацію:

— FirstName – ім'я;

— LastName – прізвище;

— PhoneNumber;

— Email –електронна адреса.

Форми з валідацією даних зображено на рисунку 4.22:

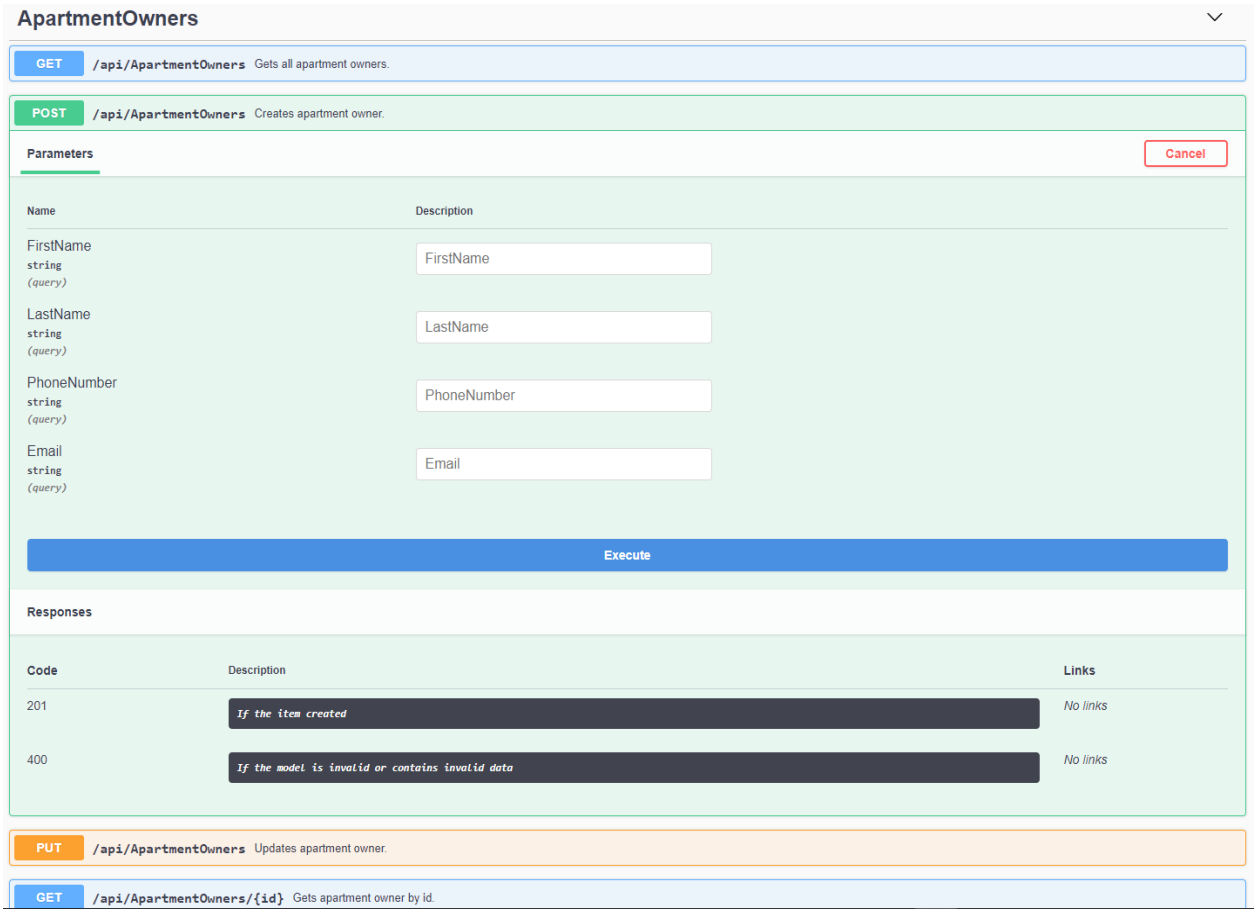

Рисунок 4.22 – Реєстрація нового власника

Кінцевим результатом запиту, може бути два варіанти відповіді системи. Якщо все пройшло гладко, система поверне код 200. Якщо було введено невірні дані, які не

пройшли перевірку, система поверне код 400, але з повідомленням про те, у якому саме місці знаходиться проблема. Як варіант може бути повідомлення про неправильне введення імені, якщо поле залишиться пустим, або якщо поле телефонного номеру буде заповнене лише цифрами від 0 до 9.

Також необхідно пам’ятати про можливість зміни сутності користувача чи видалення даних про нього. Такі операції має можливість виконувати лише адміністратор, оскільки для видалення користувача необхідний лише унікальний ідентифікатор. Система проведе пошук по базі даних та зробить видалення. Приклад роботи даної функції зображно на рисунку 4.23:

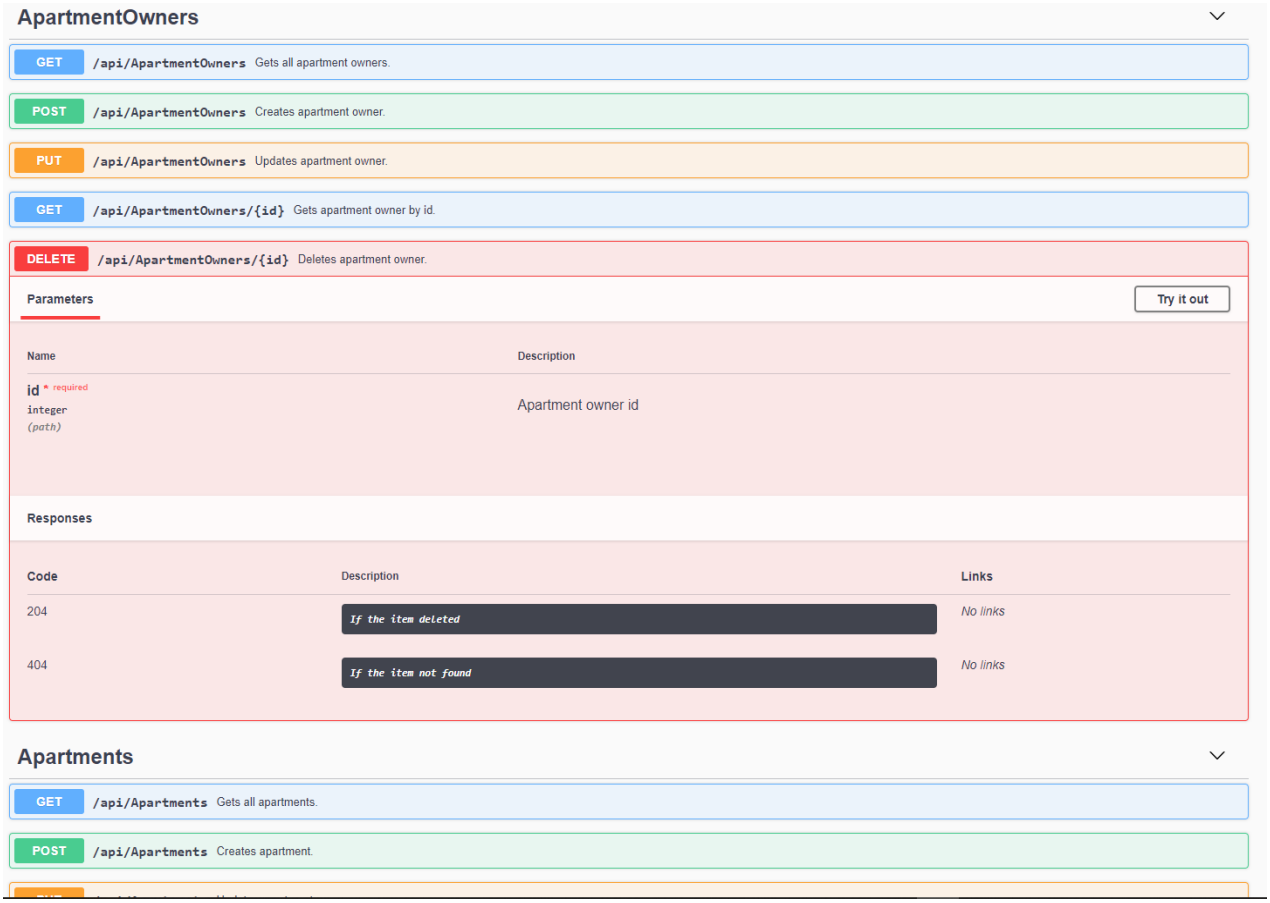

Рисунок 4.23 – Видалення власника

Далі розглянуто механізм додання до своїх активів квартири. Для цього потрібно заповнити певні поля, такі як: адреса, кількість кімнат, поверх, площа та її опис. Опис потрібен для того щоб показати кращі сторони об'єкту, які неможливо стандартизовано ввести до системи, а саме – інфраструктуру, технічне обладнання та

іншу додаткову інформацію (рисунку 4.24):

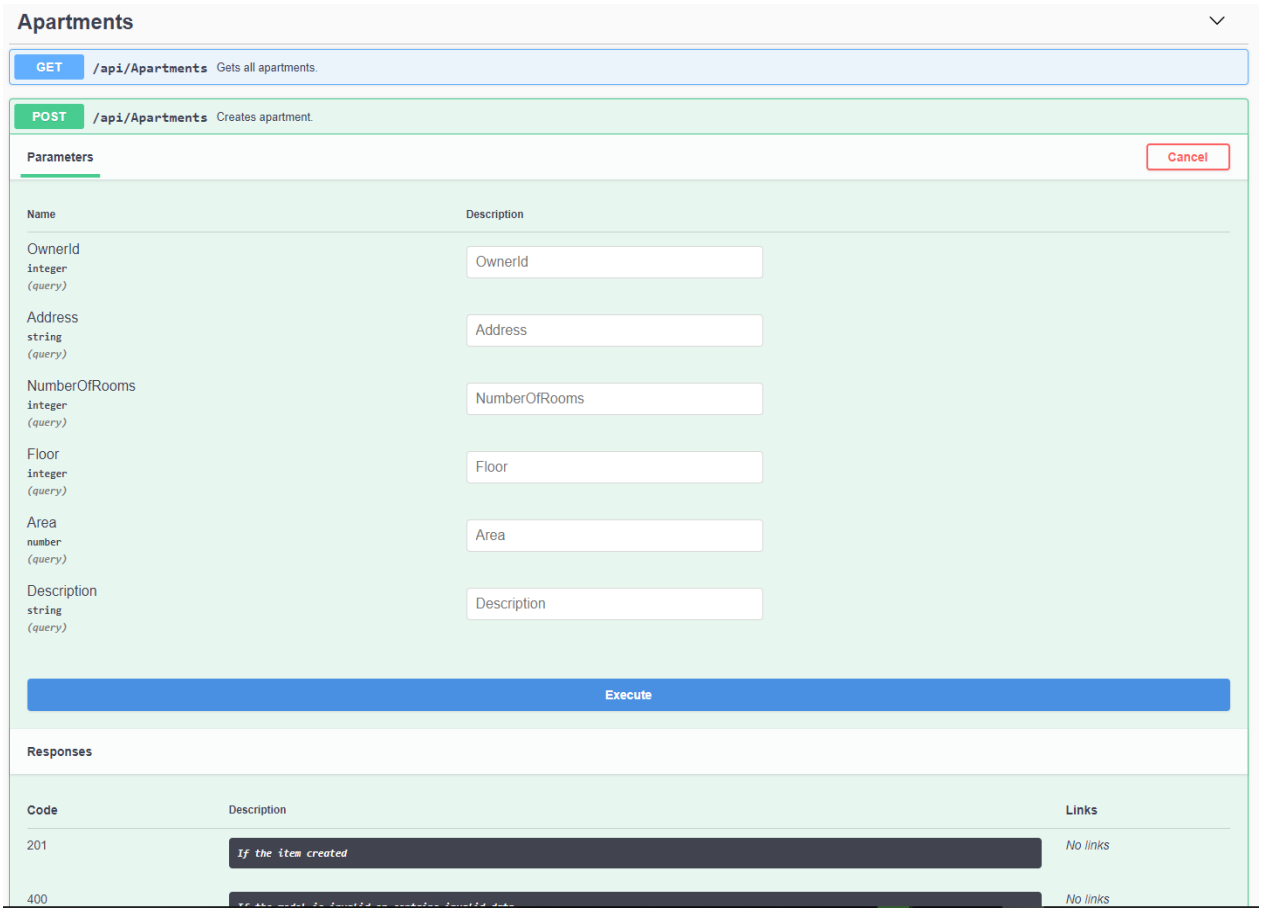

Рисунок 4.24 – Додавання квартири

Після того як за користувачем в системі зареєстровано квартиру, він може розміщати оголошення про надання приміщення в оренду. Для створення такого оголошення користувачу необхідно вказати:

—ApartmentId – унікальний ідентифікатор квартири;

—OwnerId – унікальний ідентифікатор власника;

—Rent – вартість оренди;

—TermInDays – кількість днів, на які власник готовий сдавати свою квартиру.

4.9 Тестовий приклад роботи бази даних

Побудована база даних основана на реляційній моделі даних, яка складається з сутностей. Вся інформація зберігається у вигляді таблиць. На рисунках 4.25, 4.26,4.27

наведено приклади:

	Id	AccountId	FirstName	LastName	PhoneNumber	Email	Balance	PrivateKey
1	1	1	Nikolai	Dzymidzei	88005553535	ne.mogna@gmail.com	123	4C040B0C6DD986D2
2	2	2	Viktoriya	Rozumskaya	380976543210	golova68@gmail.com	234	3F53EBE0738E2CF14
3	3	3	Egor	Nikolaev	380959116923	goranikonov@gmail.com	456	D32C34D578FAC765

Рисунок 4.25 – Таблиці сутності ApartmentOwner

	Id	ApartmentId	ApartmentOwnerId	Status	CreationDate	Discriminator	Rent	TermInDays	Price
1	1	1	1	On review	2018-01-15 00:00:00.0000000	SaleAnnouncement	NULL	NULL	2920000.00
2	2	2	2	On review	2018-02-16 00:00:00.0000000	SaleAnnouncement	NULL	NULL	2050000.00
3	3	3	3	On review	2018-03-17 00:00:00.0000000	SaleAnnouncement	NULL	NULL	1800000.00
4	4	4	4	On review	2018-04-18 00:00:00.0000000	RentAnnouncement	9000.00	180	NULL
5	5	5	5	On review	2018-05-19 00:00:00.0000000	RentAnnouncement	14000.00	365	NULL
6	6	1	1	On review	2019-05-31 20:13:09.3998672	RentAnnouncement	5000.00	10	NULL

Рисунок 4.26– Таблиця сутності Announcements

	Id	ApartmentOwnerId	Address	NumberOfRooms	Floor	Area	Description
1	1	1	Kiev, Kudryashova st. 20	2	7	62	Proper repair, with furniture and appliances, air c...
2	2	2	Kiev, Turgenevskaya st. 48	3	3	74,5	Gas column, balcony, needs repair.
3	3	3	Kiev, Zakrevskogo st. 95	3	6	80	Floor heating, fireplace (electric), direct transport ...
4	4	4	Kiev, Vasilkovskaya st. 23/16	2	2	57	Built-in kitchen, stove Bosch, hood Cata. The ba...
5	5	5	Kiev, Timoshenko st. 13a	3	18	85	

Рисунок 4.27– Таблиця сутності Apartments

4.10 Висновки до розділу 4

Оцінивши час, наданий на реалізацію дипломної роботи, була проведена декомпозиція завдань та складений план роботи. В результаті отримано програмний продукт, веб-застосунок для оренда та продажу нерухомості із вбудованою системою безпеки. Через архітектуру проекту унеможливлено вплив SQL-ін’єкцій. За допомогою системи ролей да доступів втілено модель білого списку. За допомого алгоритму шифрування Rijndael розроблено метод доступу до приватної інформації користувача.

Для того, щоб побудувати систему захисту веб-застосунку, необхідно мати на увазі технології, за допомогою яких створюється продукт та регіон в якому встановлено сервери з розгорнутим конкретним сервісом або додатком. Способи

захисту веб-додатків від зловмисників залежить від конкретних технологій і компонент, що використовуються при створенні веб-додатків, їх переваг та недоліків у даній сфері. Існують різні класифікації вразливостей, кожна атака через вразливість має свої особливості, але основними причинами вразливостей в захисті веб-додатків є помилки в розробці, реалізації та застосуванні компонентів веб- додатків. Так випливає необхідність пошуку вразливостей і реагування на інформацію про їх розташування. Як в Україні, так і в інших країнах світу створено форуми з реагування на подібні ситуації, які складаються з експертів і дослідників.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Sankar R. Burpsuite – A Beginner’s Guide For Web Application Security or Penetration Testing [Електронний ресурс] / Ravi Sankar. – 2018. – Режим доступу до ресурсу: https://kalilinuxtutorials.com/burpsuite/.

2. Ricca F. https://dl.acm.org/citation.cfm?id=381476 [Електронний ресурс] / F. Ricca, P. Tonella. – 2001. – Режим доступу до ресурсу: https://dl.acm.org/citation.cfm?id=381476

3. Ganore P. What Is A Web Server And How Does It Function? [Електронний ресурс] / Pravin Ganore. – 2017. – Режим доступу до ресурсу: https://www.milesweb.com/blog/hosting/web-server-function/.

4. How a Web server functions? [Електронний ресурс]. – 2006. – Режим доступу до ресурсу: https://www.eukhost.com/blog/webhosting/how-a-web-server- functions/.

5. Что такое веб-сервер [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://developer.mozilla.org/ru/docs/Learn/%D0%A7%D1%82%D0%BE_%D1%82%D0%B0%D0%BA%D0%BE%D0%B5_%D0%B2%D0%B5%D0%B1_%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80.

6. Корнієнко Б.Я. Дослідження імітаційного полігону захисту критичних інформаційних ресурсів методом IRISK. Моделювання та інформаційні технології. 2018. Вип. 83. С. 34-41.

7. Корнієнко Б.Я. Побудова та тестування імітаційного полігону захисту критичних інформаційних ресурсів. Наукоємні технології. 2017. № 4 (36). С. 316 - 322.

8. Korniyenko B., Yudin A., Galata L. Risk estimation of information system.Wschodnioeuropejskie Czasopismo Naukowe. 2016. № 5. P. 35 - 40.

9. Web Server and its Types of Attacks [Електронний ресурс]. – 2012. – Режим доступу до ресурсу: https://www.greycampus.com/opencampus/ethical- hacking/web-server-and-its-types-of-attacks.

10. Brewer J. Web Server Vulnerabilities and a Defense in Depth Strategy Using the

Squid Proxy [Електронний ресурс] / Jim Brewer // GSEC Practical version 1.4b. – 2004. – Режим доступу до ресурсу: https://www.giac.org/paper/gsec/3729/web-server-vulnerabilities-defense-in-depth- strategy-squid-proxy/105970.

11. 6 Common Website Security Vulnerabilities [Електронний ресурс] // Common places. – 2019. – Режим доступу до ресурсу: https://www.commonplaces.com/blog/6-common-website-security-vulnerabilities/.

12. Web Server Vulnerabilities Attacks: How to Protect Your Organization [Електронний ресурс] // Tech Funnel. – 2018. – Режим доступу до ресурсу: https://www.techfunnel.com/information-technology/web-server-vulnerabilities- attacks-how-to-protect-your-organization/.

13. Уязвимости веб-приложений [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://www.ptsecurity.com/upload/corporate/ru- ru/analytics/Web-Vulnerabilities-2019-rus.pdf.

14. Melnick J. Top 10 Most Common Types of Cyber Attacks [Електронний ресурс] / Jeff Melnick. – 2018. – Режим доступу до ресурсу: https://blog.netwrix.com/2018/05/15/top-10-most-common-types-of-cyber-attacks/

15. Корнієнко Б.Я., Юдін О.К., Снігур О.С.Безпека аутентифікації у web-ресурсах. Захист інформації. 2012. № 1 (54). С. 20 -25. DOI: 10.18372/2410-7840.14.2056 (ukr).

16. Корнієнко Б.Я., Максімов Ю.О., Марутовська Н.М. Прикладні програми управління інформаційними ризиками. Захист інформації. 2012. № 4 (57). С. 60 – 64. DOI: 10.18372/2410-7840.14.3493 (ukr).

17. Top 8 Network Attacks by Type in 2017 [Електронний ресурс] // CALYPTIX. – 2017. – Режим доступу до ресурсу: https://www.calyptix.com/top- threats/top-8-network-attacks-type-2017/.

18. Евтеев Д. SQL Injection от А до Я [Електронний ресурс] / Дмитрий Евтеев. – 2008. – Режим доступу до ресурсу: https://www.ptsecurity.com/upload/corporate/ru-ru/analytics/PT-devteev-Advanced- SQL-Injection.pdf.

19. SQL инъекции. Проверка, взлом, защита [Електронний ресурс] // BVN2. – 2011. – Режим доступу до ресурсу: https://habr.com/ru/post/130826/.

20. How to Prevent SQL Injection Attacks [Електронний ресурс] // eSecurityPlanet. – 2018. – Режим доступу до ресурсу: https://www.esecurityplanet.com/threats/how-to-prevent-sql-injection-attacks.html.

21. How to Protect Against SQL Injection Attacks [Електронний ресурс] // UC Berkeley. – 2019. – Режим доступу до ресурсу: https://security.berkeley.edu/education-awareness/best-practices-how-articles/system- application-security/how-protect-against-sql.

22. Корнієнко Б.Я. Безпека інформаційно-комунікаційних систем та мереж. Навчальний посібник для студентів спеціальності 125 «Кібербезпека». – К.:НАУ, 2018. – 226 с.

23. Корнієнко Б.Я., Галата Л.П. Оптимізація системи захисту інформації корпоративної мережі. Математичне та комп'ютерне моделювання. Серія: Технічні науки, Випуск 19, 2019. - С. 56-62.

24. SQL_Injection_Prevention_Cheat_Sheet [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.md.

25. Choudhary A. SQL Injection Attacks: Know How to Prevent Them [Електронний ресурс] / Archana Choudhary // Security Zone. – 2019. – Режим доступу до ресурсу: https://dzone.com/articles/sql-injection-attacks-know-how-to- prevent-them.

26. Cobb M. Cross-site scripting explained: How to prevent XSS attacks [Електронний ресурс] / Michael Cobb // 2009 – Режим доступу до ресурсу: https://www.computerweekly.com/tip/Cross-site-scripting-explained-How-to- prevent-XSS-attacks.

27. Singh S. 5 Practical Scenarios for XSS Attacks [Електронний ресурс] / Satyam Singh // Pentest Tools. – 2018. – Режим доступу до ресурсу: https://pentest-tools.com/blog/xss-attacks-practical-scenarios/.

28. Cross-site Scripting (XSS) [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https://www.owasp.org/index.php/Cross-site_Scripting_(XSS).

29. Cross Site Scripting (XSS) Attack Tutorial With Examples, Types & Prevention

[Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://www.softwaretestinghelp.com/cross-site-scripting-xss-attack-test/.

30. Excess XSS [Електронний ресурс]. – 2016. – Режим доступу до ресурсу: https://excess-xss.com.

31. Galata, L., Korniyenko, B., Yudin, A.: Research of the simulation polygon for the protection of critical information resources. In: CEUR Workshop Proceedings, Information Technologies and Security, Selected Papers of the XVII International Scientific and Practical Conference on Information Technologies and Security (ITS 2017), 30 Nov 2017, Kyiv, Ukraine. vol. 2067. pp. 23–31., urn:nbn:de:0074-2067-8.

32. Korniyenko B., Galata L. Implementation of the information resources protection based on the CentOS operating system. Conference Proceedings of 2019 IEEE 2nd Ukraine Conference on Electrical and Computer Engineering (UKRCON -2019) July 2 – 6, 2019, Lviv, Ukraine. - pp. 1007-1011.

33. Cross Site Scripting (XSS) Attack Tutorial With Examples, Types & Prevention [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://www.softwaretestinghelp.com/cross-site-scripting-xss-attack-test/.

34. Методы защиты от CSRF-атаки [Електронний ресурс]. – 2016. – Режим доступу до ресурсу: https://habr.com/ru/post/318748/.

35. Cross-Site Request Forgery (CSRF) [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF).

36. Testing for CSRF (OTG-SESS-005) [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://www.owasp.org/index.php/Testing_for_CSRF_(OTG-SESS-005).

37. Testing for CSRF (OTG-SESS-005) [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://www.owasp.org/index.php/Testing_for_CSRF_(OTG-SESS-005).

38. Галата Л.П., Корнієнко Б.Я., Заболотний В.В. Математична модель протидії загрозам у системі захисту критичних інформаційних ресурсів. Наукоємні технології, Том 43, № 3, 2019. – С. 300 – 306.

39. Корнієнко Б.Я. Modeling of information security system in computer network. Безпека інформаційних систем і технологій, Том №1 (1), 2019. – С.36-41.

40. Broken Authentication and Session Management [Електронний ресурс]. – 2010. – Режим доступу до ресурсу: https://www.owasp.org/index.php/Broken_Authentication_and_Session_Managemen t.

41. Charan H. Broken Authentication and Session Management—part I [Електронний ресурс] / Hari Charan. – 2017. – Режим доступу до ресурсу: https://medium.com/@grep_security/broken-authentication-and-session- management-part- i-50e760c9f599.

42. Charan H. Broken Authentication and Session Management [Електронний ресурс] / Hari Charan // DZone. – 2017. – Режим доступу до ресурсу: https://dzone.com/articles/broken-authentication-and-session-management- part.

43. Blazquez D. Broken Authentication OWASP Top 10 - A2 [Електронний ресурс] / Daniel Blazquez. – 2019. – Режим доступу до ресурсу: https://hdivsecurity.com/owasp-broken-authentication.

44. Authentication Hacking: What are Authentication Hacking Attacks? [Електронний ресурс]. – 2014. – Режим доступу до ресурсу: https://www.acunetix.com/websitesecurity/authentication/.

45. Korniyenko B., Galata L., Ladieva L. Research of Information Protection System of Corporate Network Based on GNS3. Conference Proceedings of 2019 IEEE Inernational Conference on Advanced Trends in Information Theory (IEEE ATIT -2019) Dezember 18 – 20, 2019, Kyiv, Ukraine. - pp. 244-248.

46. Korniyenko B., Galata L., Ladieva L. Mathematical model of threats resistance in the critical information resources protection system. CEUR Workshop Proceedings, Selected Papers of the XIX International Scientific and Practical Conference "Information Technologies and Security" (ITS 2019) Kyiv, Ukraine, November 28, 2019. Vol-2577. P.281-291.

47. Zeeshan N. 7 Ways To Stop Web Attacks Affecting Your Web Application [Електронний ресурс] / Nasrumminallah Zeeshan. – 2017. – Режим доступу до ресурсу: https://www.peerlyst.com/posts/7-ways-to-stop-web-attacks- affecting-your-web-

application-nasrumminallah-zeeshan.

48. Top 10-2017 A6-Security Misconfiguration [Електронний ресурс]. – 2017. – Режим доступу до ресурсу: https://www.owasp.org/index.php/Top_10- 2017_A6-Security_Misconfiguration.

49. Security Misconfiguration [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://bounty.github.com/classifications/security- misconfiguration.html.

50. TAMMANY J. https://www.sitelock.com/blog/owasp-top-10-sensitive- data-exposure/ [Електронний ресурс] / JOYCE TAMMANY // CYBER ATTACKS. – 2018. – Режим доступу до ресурсу: https://www.sitelock.com/blog/owasp-top-10-sensitive-data-exposure/.

51. Blazquez D. Sensitive Data Exposure OWASP Top 10 - A3 [Електронний ресурс] / Daniel Blazquez – Режим доступу до ресурсу: https://hdivsecurity.com/owasp-sensitive-data-exposure.

52. Корниенко Б.Я. Кибернетическая безопасность – операционные системы и протоколы. ISBN 978-3-330-08397-4, LAMBERT Academic Publishing, Saarbrucken, Deutschland. − 2017. – 122 с.

53. Korniyenko B.Y., Galata L.P. Design and research of mathematical model for information security system in computer network. Науковий журнал «Наукоємні технології». – 2017, № 2 (34), С. 114 - 118.

54. Security Testing - Sensitive Data Exposure [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://www.tutorialspoint.com/security_testing/testing_sensitive_data_exposure.htm.

55. Web Application Risk – the Threat of and Solution to Sensitive Data Exposure [Електронний ресурс] – Режим доступу до ресурсу: https://www.immuniweb.com/blog/OWASP-sensitive-data-exposure.html.

56. Using Burp to Test for Sensitive Data Exposure Issues [Електронний ресурс] // PortSwigger. – 2018. – Режим доступу до ресурсу: https://support.portswigger.net/customer/portal/articles/1965730-using-burp-to-test- for-sensitive-data-exposure-issues.

57. Common Vulnerability Scoring System v3.0 [Електронний ресурс] – Режим

доступу до ресурсу: https://www.first.org/cvss/cvss-v30-specification- v1.7.pdf.

58. Common Vulnerability Scoring System v3.0: Specification Document [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://www.first.org/cvss/specification-document.

59. Common Vulnerability Scoring System Calculator Version 3 [Електронний ресурс] – Режим доступу до ресурсу: https://nvd.nist.gov/vuln- metrics/cvss/v3-calculator.

60. Common Vulnerability Scoring System v3.0: Specification Document [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://www.first.org/cvss/specification-document.

61. Korniyenko B., Galata L., Kozuberda O. Modeling of security and risk assessment in information and communication system. Sciences of Europe. – 2016. – V. 2. – No 2 (2). – P. 61 -63.

62. Korniyenko B. The classification of information technologies and control systems. International scientific journal. – 2016. –№ 2. – P. 78 - 81.

63. Safonov L. APTSimulator — тестирование противодействия APT угрозам [Електронний ресурс] / Luka Safonov. – 2018. – Режим доступу до ресурсу: https://habr.com/ru/post/350066/.

64. Cloudi. APT SIMULATOR [Електронний ресурс] / Cloudi. – 2018. – Режим доступу до ресурсу: https://hydrasky.com/network-security/kali-tools/apt- simulator/.

65. Luka S. Обзор площадки для тестирования веб-уязвимостей OWASP Top-10 на примере bWAPP [Електронний ресурс] / Safronov Luka. – 2015. – Режим доступу до ресурсу: https://habr.com/ru/post/250551/.

66. Инструменты Kali Linux [Електронний ресурс] – Режим доступу до ресурсу: https://kali.tools.

67. Alyshov O. bWAPP Веб безопасность [Електронний ресурс] / Orkhan Alyshov – Режим доступу до ресурсу: https://orkhanalyshov.com/blog/42.

68. Using Burp Proxy [Електронний ресурс] // 2018 – Режим доступу до ресурсу: https://support.portswigger.net/customer/portal/articles/1783119-using- burp-proxy.

69. Nessus professional benefits [Електронний ресурс]. – 2019. – Режим доступу

до ресурсу: https://www.tenable.com/products/nessus/nessus-professional.

70. Корниенко Б.Я. Информационная безопасность и технологии компьютерных сетей : монография. ISBN 978-3-330-02028-3, LAMBERT Academic Publishing, Saarbrucken, Deutschland. − 2016. – 102 с.

71. Корнієнко Б.Я. Інформаційні технології оптимального управління виробництвом мінеральних добрив :монографія. – К.: Вид-во Аграр Медіа Груп, 2014. – 288 с.

72. Korniyenko B., Galata L., Ladieva L. Security Estimation of the Simulation Polygon for the Protection of Critical Information Resources / B. Korniyenko, //CEUR Workshop Proceedings, Selected Papers of the XVIII International Scientific and Practical Conference "Information Technologies and Security" (ITS 2018) Kyiv, Ukraine, November 27, 2018, Vol-2318, - P. 176-187, urn:nbn:de:0074-2318-4